Statistical validation of limiting similarity and negative co-occurrence null models: Extending the models to gain insights into sub-community patterns of community assembly

A Thesis Submitted to the College of
Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Plant Science
University of Saskatchewan
Saskatoon

By
Mike Lavender

Permission to use

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Plant Sciences
University of Saskatchewan
51 Campus Drive
Saskatoon, Saskatchewan, S7N 5A8

Abstract

Competition between species is believed to lead to patterns of either competitive exclusion or limiting similarity within ecological communities; however, to date the amount of support for either as an outcome has been relatively weak. The two classes of null model commonly used to assess co-occurrence and limiting similarity have both been well studied for statistical performance; however, the methods used to evaluate their performance, particularly in terms of type II statistical errors, may have resulted in the underreporting of both patterns in the communities tested. The overall purpose of this study was to evaluate the efficacy of the negative co-occurrence and limiting similarity null models to detect patterns believed to result from competition between species and to develop an improved method for detecting said patterns. The null models were tested using synthetic but biologically realistic presence-absence matrices for both type I and type II error rate estimations. The effectiveness of the null models was evaluated with respect to community dimension (number of species × number of plots), and amount of pattern within the community. A novel method of subsetting species was developed to assess communities for patterns of co-occurrence and limiting similarity and four methods were assessed for their ability to isolate the species contributing signal to the pattern. Both classes of null model provided acceptable type I and type II error rates when matrices of more than 5 species and more than 5 plots were tested. When patterns of negative co-occurrence or limiting similarity were add to all species both null models were able to detect significant pattern ($\beta > 0.95$); however, when pattern was added to only a proportion of species the ability of the null models to detect pattern deteriorated rapidly with proportions of 80% or less. The use of species subsetting was able to detect significant pattern of both co-occurrence and limiting similarity when fewer than 80% of species were contributing signal but was dependent on the metric used for the limiting similarity null model. The ability of frequent pattern mining to isolate the species contributing signal shows promise; however, a more thorough evaluation is required in order to confirm or deny its utility.

Acknowledgments

Thanks to both of my advisors, Dr Eric Lamb and Dr. Brandon Schamp, for their continued support, guidance and tolerance throughout the duration of this project as well as many thanks to my supervisory committee: Dr. Phil McLoughlin, Dr. Chris Willenborg, Dr. Yuguang Bai, and Dr. Bruce Coleman.

I would also like to acknowledge the contribution that my lab mates, in no particular order, Candace Piper, Christiane Catellier, Jenalee Mischkolz, Amand Guy, Udayanga Attanayake and Danielle Levesque for providing moral support and guidance along the way.

Thank you to the software developers Chris Tarttelin, Dave Ford, Marcos Tarruella, Aidan Rogers and James Townley for mentoring me on software development (I still have a long way to go) and offering guidance to me as I learned to program in Scala in my pre-grad school life. If it had not been for their time and patience this project would never have been able to happen.

I must also acknowledge the contributions made by the University of Saskatchewan's HPC cluster and WestGrid (Compute Canada) to this project. It was access to these computing facilities that allowed me to generate my data in a timely manner and thank you to Jason Hlady for providing the initial guidance to get me going on these resources.

I would also like to thank the Department of Plant Sciences for funding this research as well as the Natural Sciences and Engineering Research Council of Canada for providing funding through NSERC Discovery grants to both Dr. Eric Lamb and Dr. Brandon Schamp.

Finally I would like to thank my family, Monika, Seren and Dax as well as my parents for their support, tolerance, and efforts to understand what I was doing as I worked through this project.

Table of Contents

List of Figures

List of Abbreviations

AITS – Abundance independent trait shuffling

AWTS – Abundance weighted trait shuffling

NN – Mean nearest neighbour trait distance

SDNN – Standard deviation of the nearest neighbour trait distance

1.0 General Introduction

1.1 Background

1.1.1 Community assembly, co-occurrence, and limiting similarity

One of the tenets of community ecology is that there exists a set of rules that constrain community assembly (Diamond, 1975) and that these rules are the combined result of species interactions and abiotic factors. While stochastic processes are also important in community assembly (Helsen, Hermy, & Honnay, 2012; Hubbell, 2001; 2005; Rosindell, Hubbell, & Etienne, 2011), it is our ability to understand the mechanisms underpinning deterministic (non-random) assembly processes that, aside from advancing our overall understanding of community ecology, have potential for habitat restoration efforts, carrying out impact assessments, prediction of and control of invasive species, and the management of existing communities.

Although ecological communities consist of all interacting species within a given habitat most ecologists are interested in community assembly within a taxonomic level. Two concepts commonly used in the evaluation of taxonomic communities for signs of assembly rules are negative co-occurrence, and limiting similarity. Negative co-occurrence, derived from the work of Gause (1932) and his Competitive Exclusion Principle, is based on the premise that species that compete for a limiting resource will be unable to co-exist. By examining species distribution data for patterns of positive and negative co-occurrence the objective is to infer rules about which species can exist together and which cannot. Limiting similarity, based on the work of MacArthur and Levins (1964; 1967), is the idea that species are able to avoid competitive exclusion through variations in traits and that these variations in traits represent the specialized ways that species have of using resources (MacArthur & Levins, 1964). As with co-occurrence analyses, the distribution of trait values are analysed and rules are inferred from their dispersion.

Both approaches of inferring assembly rules rely on the existence of a measurable signature, in practice their signature can be obscured by factors such as natural variation in

trait values and temporal variation in resource availability potentially resulting in incomplete competitive exclusion. Either of these processes adds noise to the system under investigation. Given the number of factors that can act to obscure patterns of limiting similarity or negative co-occurrence, it is imperative that robust statistical methods be used to help determine what is noise versus real pattern (Connor & Simberloff, 1979). The statistical method commonly used for these types of analyses is a null modelling approach using two classes of null model: limiting similarity, and negative co-occurrence.

In the 30 years since Conner and Simberloff's (1979) review of Diamond's (1975) work and their concern over statistical procedures, there have been significant advances made in our ability to detect patterns of negative co-occurrence, although problems accounting for variation resulting from abiotic factors still exist. Several indices of community co-occurrence patterns have been devised (CHECKER, Diamond, 1975; COMBO, Pielou & Pielou, 1968; V ratio, Schluter & Grant, 1984; C-Score, Stone & Roberts, 1990) and the use of null models and their implementations have evolved enough to be able to provide a reasonable level of confidence in differentiating between patterns of competition versus stochastic processes (see Gotelli, 2000; Harvey, Colwell, Silvertown, & May, 1983; Ulrich & Gotelli, 2007 for a review of current approaches in null models).

Despite advances in the negative co-occurrence analyses there are still several shortcomings associated with them. Notably they are not a direct test of competition but instead are a test of patterns that may or may not be the result of competition. Another shortcoming of the co-occurrence analysis is that, despite being a measure of between species co-occurrence, they do not indicate which species pairs are driving community-scale patterns. Pairwise tests of co-occurrence have been developed (Gotelli & Ulrich, 2010; Sfenthourakis, Giokas, & Tzanatos, 2004; Veech, 2013) that may be able to provide insight in to which species pairs are driving community scale patterns; however, these methods still need to be used with caution (Gotelli & Ulrich, 2010).

Where whole community approaches to co-occurrence analysis attempt to assess the role of competition through species distribution patterns, limiting similarity analysis attempts to assess the role of competition in assembling communities through consideration of trait-based measures. Where co-occurrence tests evaluate the existence of

segregating, or negatively co-occurring species pairs against a null model, limiting similarity tests evaluate the distribution of traits against a null model. If trait values of co-existing species are more widely spaced (overdispersed) relative to the null model then this is taken as an indication of limiting similarity or the product of competition. On the contrary, if coexisting species are more similar than expected for given traits, two explanations are possible. First, it may be that trait convergence/under-dispersion relates to environmental filtering such that coexisting species share traits that are related to the ability to grow in that environment (Kraft, Valencia, & Ackerly, 2008). Second, it is possible that competition may result in less competitive species being excluded. If these species represent an end of the trait spectrum that contributes to the lower competitive ability of these species, then competition may drive convergence (Grime, 2006; MacArthur & Levins, 1967; Mayfield & Levine, 2010; Schamp, Chau, & Aarssen, 2008).

Limiting similarity tests, like co-occurrence analyses, are limited in the information that they can provide. These tests provide no specific information on how or which species are competing; however, they can identify species-level traits within the community that are under- or overdispersed. It is also possible that limiting similarity tests are sensitive to a neutralizing effect when both under- and over-dispersion occur within a community (Götzenberger et al., 2012; Schamp et al., 2008). For example, within a community it seems reasonable to assume that a subset of species will positively co-occur while a second subset will negatively co-occur. These subsets could be the result of variations in microhabitat within the community or convergent evolution of traits resulting from species-saturation (Scheffer & van Nes, 2006). If this is the case and we are correct in assuming that co-occurring species have greater variation in trait values and negatively co-occurring species have more similar trait values then, when we test for trait variation at the community level, these two subsets may cancel each other out such that community-level analyses produce no evidence of limiting similarity (Götzenberger et al., 2012). It follows that limiting similarity tests might be more informative if we first partition a community into groups of positively and negatively co-occurring species and then perform limiting similarity tests on these subsets.

By first dividing communities into two groups of species that either positively, or negatively co-occur and then running the limiting similarity null model on these groups, it may help to provide more insight into the role that limiting similarity plays by limiting analysis to groups of species that are either interacting (positively co-occurring) or not (negatively co-occurring). By removing the potential background noise in the community-wide analysis the signal, our ability to detect patterns of limiting similarity within subsets of species, should become stronger. Also, if instead of grouping species by co-occurrence, they are grouped by trait similarity or niche overlap it may be possible to determine the importance that trait values play in community assembly. That is, if a similar trait or niche value leads to greater or lesser segregation then is expected by chance. Essentially the process of sub-setting species, whether it is by trait values or measure of co-occurrence, provides a selective filter of community data that should allow for cleaner signal in both co-occurrence and limiting similarity analyses and as a result enable clearer understanding of the assembly rules that communities are subject to.

1.1.2 Statistical evaluations of null models

Both negative co-occurrence and limiting similarity null models have undergone statistical verification to ensure that they perform with acceptable Type I and Type II error rates (co-occurrence: Fayle & Manica, 2010; Gotelli, 2000; limiting similarity: Hardy, 2008); however, this testing has been confined to a limited set of matrix sizes: 20 × 20 (Gotelli, 2000), 40, 100 and 160 species (Hardy, 2008), 10 × 10 and larger (Fayle & Manica, 2010), in some cases have relied on the "sequential swap" randomisation algorithm (Fayle & Manica, 2010) which is vulnerable to serial correlation (Gotelli & Ulrich, 2011), or used biologically unrealistic matrices for type II error rate estimations (Fayle & Manica, 2010; Gotelli, 2000). Given these limitations, it is unknown if these null-model tests hold up under a broad range of community dimensions (i.e., the "area" of a rectangular matrix made up of *m* species by *n* plots).

1.2 Thesis objectives

The objectives of this study are fourfold. The first objective is to estimate the type I and type II error rates for a broad range of matrix dimensions and to do so using biologically

realistic presence-absence matrices. The second objective is to determine the statistical power of the null models with respect to the proportion of species contributing signal to the patterns of negative co-occurrence and limiting similarity. The third objective is to evaluate a novel method for detecting patterns of negative co-occurrence and limiting similarity when only a proportion of species are contributing signal to the pattern. The fourth and final objective is to assess the potential of several methods to correctly deduce the species that are contributing signal to the patterns detected by the null models.

1.2.1 Thesis overview

Chapter Two addresses the statistical limitations of the negative co-occurrence and limiting similarity null models by extending the range of matrix dimensions tested. To determine if there were minimum matrix dimensions that the null models could be used with we tested matrices as small as three × three and larger, while maintaining the same (biologically realistic) constraints of species richness and abundance for both the type I and type II error rate estimations. In Chapter Two we test how the null models perform statistically when the patterns of co-occurrence within the matrices are purely stochastic. That is, we evaluate how often falsely significant results occur when there is no pattern of negative co-occurrence or limiting similarity beyond what would occur by random chance.

Chapter Three addresses the statistical power of the null models, or the ability of the models to detect patterns of negative co-occurrence and limiting similarity when they do exist. While the null models have been used to look for patterns of both negative co-occurrence (Gotelli & Ellison, 2002; Gotelli & Rohde, 2002; Maestre & Reynolds, 2007; Maltez-Mouro, Maestre, & Freitas, 2010; Ribichich, 2005; Rooney, 2008; e.g., Weiher, Clarke, & Keddy, 1998; Zhang et al., 2009) and limiting similarity (e.g., Cornwell, Schwilk, & Ackerly, 2006; de Bello et al., 2009; Hardy, 2008; Kraft et al., 2008; Kraft & Ackerly, 2010; Mouillot, Dumay, & Tomasini, 2007; Schamp et al., 2008; Schamp & Aarssen, 2009; Stubbs & Wilson, 2004; e.g., Weiher et al., 1998; Wilson & Stubbs, 2012), the amount of support for each varies. In a meta-analysis of community assembly processes Götzenberger et al. (2012) found that only 18% of studies found significant support for limiting similarity whereas, 41% of studies found significant support for co-occurrence. These relatively low detection rates may indicate that, although the type I and II error rates are acceptable, the

statistical power of these tests is weak. While aspects of the type II error rate (and statistical power) of the co-occurrence null model have been assessed (Gotelli, 2000), no similar attempt has been made for two commonly used limiting similarity null models: Abundance Independent Trait Shuffling (AITS) or, Abundance Weighted Trait Shuffling (AWTS) although Hardy (2008) did carry out a partial analysis of each for the purposes of his study. As such Chapter Three also addresses this knowledge gap in the limiting similarity null models.

Having established the type I and type II error rates of the negative co-occurrence and limiting similarity null models in Chapter Two and the statistical power of both models in Chapter Three, Chapter Four then builds on this knowledge by developing and testing a novel method for detecting patterns of limiting similarity and negative co-occurrence. The purpose of this approach is to be able to detect patterns of co-occurrence and limiting similarity in communities when signal arises from only a proportion of the species in the community under investigation. This novel method, still utilizing null models, examines subsets of species within the community versus a whole community null modeling approach. Chapter Four then assess the ability of frequent pattern mining, indicator species analysis, Veech's (2013) probabilistic model of species co-occurrence, and the 95 percent confidence limit criterion (Sfenthourakis et al., 2004) to discover species within a community that are contributing signal to patterns of con-occurrence and limiting similarity.

1.3 References

Connor, E. F., & Simberloff, D. (1979). The assembly of species communities: Chance or competition? *Ecology*, *60*, 1132–1140.

Cornwell, W. K., Schwilk, D. W., & Ackerly, D. D. (2006). A trait-based test for habitat filtering: Convex hull volume. *Ecology*, *87*, 1465–1471.

de Bello, F., Thuiller, W., Lepš, J., Choler, P., Clément, J.-C., Macek, P., et al. (2009). Partitioning of functional diversity reveals the scale and extent of trait convergence and divergence. *Journal of Vegetation Science*, *20*, 475–486.

Diamond, J. M. (1975). Assembly of species communties. In *Ecology and evolution of communities* (pp. 342–444). Cambridge, MA, US.: Belknap Press of Harvard University Press.

Fayle, T. M., & Manica, A. (2010). Reducing over-reporting of deterministic co-occurrence patterns in biotic communities. *Ecological Modelling*, *221*, 2237–2242.

Gause, G. F. (1932). Experimental studies on the struggle for existence I. Mixed population of two species of yeast. *Journal of Experimental Biology*, *9*, 389–402.

Gotelli, N. J. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, *81*, 2606–2621.

Gotelli, N. J., & Ellison, A. M. (2002). Assembly rules for New England ant assemblages. *Oikos*, *99*, 591–599.

Gotelli, N. J., & Rohde, K. (2002). Co-occurrence of ectoparasites of marine fishes: A null model analysis. *Ecology Letters*, *5*, 86–94.

Gotelli, N. J., & Ulrich, W. (2010). The empirical Bayes approach as a tool to identify non-random species associations. *Oecologia*, *162*, 463–477.

Gotelli, N. J., & Ulrich, W. (2011). Over-reporting bias in null model analysis: A response to Fayle and Manica (2010). *Ecological Modelling*, *222*, 1337–1339.

Götzenberger, L., de Bello, F., Bråthen, K. A., Davison, J., Dubuis, A., Guisan, A., et al. (2012). Ecological assembly rules in plant communities--approaches, patterns and prospects. *Biological reviews of the Cambridge Philosophical Society*, *87*, 111–127.

Grime, J. P. (2006). Trait convergence and trait divergence in herbaceous plant communities: Mechanisms and consequences. *Journal of Vegetation Science*, *17*, 255–260.

Hardy, O. J. (2008). Testing the spatial phylogenetic structure of local communities: Statistical performances of different null models and test statistics on a locally neutral community. *Journal of Ecology*, *96*, 914–926.

Harvey, P. H., Colwell, R. K., Silvertown, J. W., & May, R. M. (1983). Null Models in Ecology. *Annual review of ecology and systematics*, *14*, 189–211.

Helsen, K., Hermy, M., & Honnay, O. (2012). Trait but not species convergence during plant community assembly in restored semi-natural grasslands. *Oikos*, EV 1–EV 11.

Hubbell, S. P. (2001). *The unified neutral theory of biodiversity and biogeography*. Princeton University Press.

Hubbell, S. P. (2005). Neutral theory in community ecology and the hypothesis of functional equivalence. *Functional Ecology*, *19*, 166–172.

Kraft, N. J. B., & Ackerly, D. D. (2010). Functional trait and phylogenetic tests of community assembly across spatial scales in an Amazonian forest. *Ecological Monographs*, *80*, 401–422.

Kraft, N. J. B., Valencia, R., & Ackerly, D. D. (2008). Functional traits and niche-based tree community assembly in an Amazonian forest. *Science (New York, N.Y.)*, *322*, 580–582.

MacArthur, R., & Levins, R. (1964). Competition, Habitat Selection, and Character Displacement in a Patchy Environment. *Proceedings of the National Academy of Sciences of the United States of America*, *51*, 1207–1210.

MacArthur, R., & Levins, R. (1967). The limiting similarity, convergence, and divergence of coexisting species. *American Naturalist*, 377–385.

Maestre, F. T., & Reynolds, J. F. (2007). Amount or pattern? Grassland responses to the heterogeneity and availability of two key resources. *Ecology*, *88*, 501–511.

Maltez-Mouro, S., Maestre, F. T., & Freitas, H. (2010). Co-occurrence patterns and abiotic stress in sand-dune communities: Their relationship varies with spatial scale and the stress estimator. *Acta Oecologica*, *36*, 80–84.

Mayfield, M. M., & Levine, J. M. (2010). Opposing effects of competitive exclusion on the phylogenetic structure of communities. *Ecology Letters*, *13*, 1085–1093.

Mouillot, D., Dumay, O., & Tomasini, J. A. (2007). Limiting similarity, niche filtering and functional diversity in coastal lagoon fish communities. *Estuarine, Coastal and Shelf Science*, *71*, 443–456.

Pielou, D. P., & Pielou, E. C. (1968). Association among species of infrequent occurrence: The insect and spider fauna of Polyporus betulinus (Bulliard) Fries. *Journal of theoretical biology*, *21*, 202–216.

Ribichich, A. M. (2005). From null community to non-randomly structured actual plant assemblages: Parsimony analysis of species co-occurrences. *Ecography*, *28*, 88–98.

Rooney, T. P. (2008). Comparison of co-occurrence structure of temperate forest herb-layer communities in 1949 and 2000. *Acta Oecologica*, *34*, 354–360.

Rosindell, J., Hubbell, S. P., & Etienne, R. S. (2011). The unified neutral theory of biodiversity and biogeography at age ten. *Trends in Ecology \& Evolution*, *26*, 340–348.

Schamp, B. S., & Aarssen, L. W. (2009). The assembly of forest communities according to maximum species height along resource and disturbance gradients. *Oikos*, *118*, 564–572.

Schamp, B. S., Chau, J., & Aarssen, L. W. (2008). Dispersion of traits related to competitive ability in an old-field plant community. *Journal of Ecology*, *96*, 204–212.

Scheffer, M., & van Nes, E. H. (2006). Self-organized similarity, the evolutionary emergence of groups of similar species. *Proceedings of the National Academy of Sciences*, *103*, 6230–6235.

Schluter, D., & Grant, P. R. (1984). Determinants of morphological patterns in communities of Darwin's Finches. *The American naturalist*, *123*, 175–196.

Sfenthourakis, S., Giokas, S., & Tzanatos, E. (2004). From sampling stations to archipelagos: Investigating aspects of the assemblage of insular biota. *Global Ecology and Biogeography*, *13*, 23–35.

Stone, L., & Roberts, A. (1990). The checkerboard score and species distributions. *Oecologia*, *85*, 74–79.

Stubbs, W. J., & Wilson, J. B. (2004). Evidence for limiting similarity in a sand dune community. *Journal of Ecology*, *92*, 557–567.

Ulrich, W., & Gotelli, N. J. (2007). Disentangling community patterns of nestedness and species co-occurrence. *Oikos*, *116*, 2053–2061.

Veech, J. A. (2013). A probabilistic model for analysing species co-occurrence. *Global Ecology and Biogeography*, *22*, 252–260.

Weiher, E., Clarke, G. D. P., & Keddy, P. A. (1998). Community assembly rules, morphological dispersion, and the coexistence of plant species. *Oikos*, *81*, 309–322.

Wilson, J. B., & Stubbs, W. J. (2012). Evidence for assembly rules: Limiting similarity within a saltmarsh. *Journal of Ecology*, *100*, 210–221.

Zhang, J., Hao, Z., Song, B., Li, B., Wang, X., & Ye, J. (2009). Fine-scale species co-occurrence patterns in an old-growth temperate forest. *Forest Ecology and Management*, *257*, 2115–2120.

Chapter 2.0 Preamble

This chapter establishes the baseline statistical performance of the negative co-occurrence and limiting similarity null models (i.e. the type I and type II error rates) across a broad range of matrix dimensions and relates to the overall thesis by assessing two specific aspects of the type I and type II error rate estimations: the influence of matrix dimension on error rates with a focus on matrices smaller than those previously tested, and to determine the type II error rate estimates for the co-occurrence and limiting similarity null models when biologically realistic matrices are used for the estimations. The results of this chapter indicate that both classes of null model perform with acceptable type I and type II error rates, that the models should be used with matrices that have six or more species and that exclusive *p*-values result in better type I error rates without a substantial increase in type II error rates.

This chapter relates to the overall thesis by addressing the first objective which was to determine the type I and type II error rates of the null models with respect to matrix dimension and to do so using biologically realistic matrices for both types of error rate estimations.

2.0 Statistical limitations of co-occurrence and limiting similarity null models: effects of matrix dimension and *p*-value calculation.

2.1 Abstract

The use of null models for species co-occurrence and limiting similarity analyses have become standard practice in community ecology; however, no detailed assessment has been made of the stability of these models when the number of species and/or plots falls below 20. To determine the statistical performance of these models with respect to matrix dimension (species × plots), presence-absence matrices ranging in size from 3 to 50 species and 3 to 50 plots were randomly generated. Species abundances were distributed log-normally and placement within plots was random. Trait values for limiting similarity tests were drawn randomly from a uniform distribution. The C-Score test statistic and "fixed-fixed" independent swap algorithm were used for co-occurrence null models. For limiting similarity null models, we used the mean Nearest Neighbour (NN) trait distance and the Standard Deviation of Nearest Neighbour distances (SDNN) as test statistics, and considered two randomization algorithms: abundance independent trait shuffling (AITS) and, abundance weighted trait shuffling (AWTS). Matrices as small as three × three resulted in acceptable type I error rates ($p < 0.05$) for both the co-occurrence and limiting similarity null models when exclusive *p*-values were used. The use of inclusive (≤ or ≥) versus exclusive (< or >) *p*-values increased type I error rates particularly for smaller dimensioned matrices. This appears to be due to the fact that randomization of small matrices more frequently produces the same matrix, and the resolution of the C-Score index decays with decreasing matrix dimension. Exclusive *p*-values should be used for both the limiting similarity and co-occurrence null models in order to reduce the type I error rates without a significant increase in type II error rates. The application of both classes of null model should be restricted to matrices with five or more species to avoid the possibility of type II errors and AITS in combination with the NN metric should be used over AWTS and the SDNN metric to avoid type II errors in the limiting similarity null model.

2.2 Introduction

One of the tenets of community ecology is that there exists a set of rules that constrain community assembly (Diamond, 1975) and that these assembly rules are the combined result of species interactions and abiotic factors. In an effort to determine these rules several tools have been developed, one of which, the null model, has become commonly used. Two classes of null model, limiting similarity and negative co-occurrence, are regularly used to evaluate the influence of competition on community structure. While both classes of null model focus on linking community patterns to the competitive process, they do so in different ways. Co-occurrence null models test for patterns of species segregation across samples that may be the result of competitive exclusion. Limiting similarity null models (i.e. trait-based assembly null models) test whether species differing in traits related to niche partitioning are found growing together more commonly than expected by chance (i.e., positively associated). Trait-based null models can identify either the aggregation or segregation of species with similar traits. Aggregation is often considered as evidence of abiotic (environmental) filtering and segregation as evidence of biotic filtering (i.e., competitive filtering) of the kind predicted under limiting similarity.

Both classes of null model have undergone statistical verification to ensure that they perform with acceptable Type I and Type II error rates (Fayle & Manica, 2010; Gotelli, 2000; Hardy, 2008); however, this testing has been confined to a limited set of matrix sizes: 20 × 20 (Gotelli, 2000), 40, 100 and 160 species (Hardy, 2008), 10 × 10 and larger (Fayle & Manica, 2010), in some cases have relied on the "sequential swap" randomisation algorithm (Fayle & Manica, 2010) which is vulnerable to serial correlation (Gotelli & Ulrich, 2011), or used biologically unrealistic matrices for type II error rate estimations (Fayle & Manica, 2010; Gotelli, 2000). Given these limitations, it is unknown if these null-model tests hold up under a broad range of community dimensions (i.e., the "area" of a rectangular matrix made up of *m* species by *n* plots).

In the most comprehensive analysis of co-occurrence null models and their indices, a single matrix dimension was used for Type I error rate estimations (17 species × 19 plots) and Type II error rate estimations (20 species × 20 plots) (Gotelli, 2000). Although the matrix dimensions were limited in range, Gotelli (2000) has to date provided the most

thorough analysis of co-occurrence null models and as a result is often cited as justification for selecting randomisation algorithms and co-occurrence metrics. Fayle and Manica (2010) extended the range of dimensionality covered for these types of analyses; however the randomisation algorithm they tested was sub-optimal (Gotelli, 2000; Gotelli & Ulrich, 2011), did not extend error rate estimations for matrices with fewer than 100 cells (a 10 × 10 matrix), and as with the work of Gotelli (2000), relied on biologically unrealistic matrices for type II error rate estimations.

The use of biologically unrealistic matrices for the type II error rate estimations, a problem with both Gotelli's (2000) and Fayle and Manica's (2010) work, is likely to result in incorrect error rate estimations for the models tested. Whether these matrices are likely to result in over or under reporting of error rates is unknown. The matrices used for the type I error rate estimates conform to the expectation of a log-normal species abundance distribution (Ulrich, Ollik, & Ugland, 2010), yet this constraint is lost for the type II estimates. While the matrices used for type II estimations do contain known amounts and types of pattern, the applicability of the results to real community data is uncertain.

Another unresolved issue with co-occurrence null models is how well they perform at the lower end of matrix dimensionality. Reduction of matrix size through small numbers of species or plots can be expected to adversely affect these null models as it reduces both the number of ways that the matrix can be shuffled and the granularity of the C-Score values. For example, a three × three matrix can be shuffled a maximum of three ways using the independent swap algorithm and has a granule size equal to 1/3. Granule size is equal to one checkerboard unit divided by the number of pairwise comparisons $1/(r(r-1)/2)$ where *r* is the number of rows in the matrix. This value, the minimum incremental change in C-Score, combined with the number of plots and matrix density determine the range of possible C-Score values. It is unknown if constraints on this range may influence the results.

The efficacy of null models on mid to large sized matrices is well established (Fayle & Manica, 2010; Gotelli, 2000; Gotelli & Ulrich, 2011), however the number of studies that have used the co-occurrence null model below the tested matrix dimension of 20 × 20 (Gotelli, 2000) suggests that validation as a tool for small matrices is needed (Burns, 2007;

Gainsbury & Colli, 2003; Gotelli & McCabe, 2002; e.g., Gotelli & Rohde, 2002; Heino, 2013; Mouillot, Dumay, & Tomasini, 2007a). In summary, the conclusions of a large number of studies rely on the premise that error rates for co-occurrence null model tests remain stable across a wide range of community dimensions.

The situation for limiting similarity is similar to that of co-occurrence in that performance of these null models across a breadth of community dimensions remains uncertain. While Hardy (2008) varied matrix size in his assessment of error rates he did so only in one dimension (species) and for a limited range of values (40, 100 and 160 species). While the typical size of matrices collected by ecologists is encompassed by this range (species < 100) (Gotelli & Ulrich, 2011), this is not always the case (e.g., Stubbs & Wilson 2004 species = 9; Schamp *et al.* 2008 species < 13; Kraft & Ackerly 2010 species = 1,083; Baraloto *et al.* 2012 species = 499; Wilson & Stubbs 2012 species = 11 & 14). Despite Hardy's (2008) work, it is rarely cited as the rationale for model selection (but see Wilson & Stubbs, 2012), nor have many authors cited other relevant studies or attempted themselves to address the statistical performance of their chosen null models (Mayfield, Boni, Daily, & Ackerly, 2013; Smith, Moore, & Grove, 1994; Wilson, 1989) (but see Kraft & Ackerly, 2010). As such, our understanding of the general trends across limiting similarity studies (Götzenberger et al., 2012) may be hampered by problems related to the error rates for particular focal matrix sizes. These potential error rate related problems may explain why support for limiting similarity has been inconsistent with studies finding both support for limiting similarity (Kraft, Valencia, & Ackerly, 2008; Stubbs & Wilson, 2004; e.g., Weiher, Clarke, & Keddy, 1998; Wilson & Stubbs, 2012) and against (Baraloto et al., 2012; Dante, Schamp, & Aarssen, 2013; e.g., Franzén, 2004; Mouillot, Mason, & Wilson, 2007b; Mouillot et al., 2005; Schamp & Aarssen, 2009; Schamp, Chau, & Aarssen, 2008; Schamp, Hettenbergerová, & Hájek, 2011; Schamp, Horsák, & Hájek, 2010) and why, in a recent review, Gotzenberger (2012) found little evidence of limiting similarity among the studies considered.

As the number of studies using either co-occurrence or limiting similarity null models increases (Götzenberger et al., 2012), it remains uncertain as to the suitability of these tests across a broad range of matrix dimensions that is, there still exist no clear

guidelines on matrix dimension that are suitable for these types of analyses. In this study, we sought to alleviate this uncertainty by determining the statistical performance, type I and II error rates, of each model across a broader range of community dimensions (matrix sizes) than has been done to date.

2.3 Materials and methods

2.3.1 Generating presence-absence matrices

The starting point for the type I and II error rate estimations consisted of generating presence-absence matrices of *m* rows and *n* columns where each row represented a species and each column a plot (or site). Species presence within a plot was represented by a '1' and absence by '0'.

Matrices of *m* rows by *n* columns were generated by randomly assigning individuals to a matrix cell using the following method. The incidence, N_i, of each species (the number of plots that a species occurred in) was determined by sampling from a log-normal distribution ($N_i = e^{x_i/2a}$) where $x_i \sim N(0, 1)$ and a is the shape generating parameter (Ulrich & Gotelli, 2010). The shape-generating parameter a was set to 0.2 as that value is appropriate for large, well sampled communities (Preston, 1962a; 1962b). The log-normal distribution was chosen over the power series and log series distributions due to its combined fit to well sampled and incompletely sampled communities (Ulrich et al., 2010). The resulting incidence values were converted to integers (from decimal values) by truncation and values equal to zero or greater than the number of plots (n) were discarded. This was done as presence-absence data are binary in nature and row totals, logically, could only be integer values. The removal of zero values was required to prevent the generation of matrices with empty rows (degenerate matrices: Gotelli 2000).

Species were assigned to plots with the probability of assignment proportional to the number of incidences of that species divided by the number of plots. Species presences were assigned this way until all incidences were accounted for and all plots had at least one species present. If the resulting matrix contained a plot with no species present, the process was repeated. This resulted in matrices that had at least one occurrence for each species and at least one species present in each plot. A similar approach to matrix

generation was used by Ulrich and Gotelli (2010) to generate ecologically realistic abundance data. Other approaches, such as a neutral model, could have been used to generate the community data (Bell, 2000); however, our approach does not presuppose an underlying ecological mechanism and is consistent with previous approaches (Gotelli, 2000; Ulrich & Gotelli, 2010). The resulting communities contained log-normal species abundance patterns such as those commonly observed in natural systems.

To determine if species relative abundances influenced co-occurrence null models three indices were calculated for 1000 randomly generated 50 × 50 matrices; Simpson's Index of Evenness, Shannon's Index of Evenness and the slope of the log-transformed rank abundance curve. Indices were plotted against co-occurrence null model SES values and a Pearson product-moment correlation was performed for each index.

2.3.2 Null models

A single null model, the fixed-fixed independent swap algorithm (Connor & Simberloff, 1979; Gotelli, 2000), in combination with the C-Score (Stone & Roberts, 1990) was used for all co-occurrence analyses. This algorithm and metric were chosen due to their prevalence in the literature and previous validation as a reliable measure of co-occurrence (Gotelli, 2000). For limiting similarity tests, two null model randomization methods were used in combination with two metrics: mean Nearest Neighbour (NN) distances and, Standard Deviation of Nearest Neighbour distances (SDNN). The two null models were Abundance-Independent Trait Shuffling (AITS) and, Abundance-Weighted Trait Shuffling (AWTS) (Dante et al., 2013). The AITS method of shuffling has been commonly employed in previous studies (Cornwell & Ackerly, 2013; Dante et al., 2013; e.g., Kraft et al., 2008; Schamp & Aarssen, 2009) and consists of shuffling trait values between species without constraint. The AWTS null model is an alternative method of shuffling trait values between species that preserves trait abundance and attempts to separate trait abundance from trait distribution within the community (Dante et al., 2013; de Bello et al., 2009; Hardy, 2008; Kraft et al., 2008; e.g., Stubbs & Wilson, 2004; Wilson & Stubbs, 2012). We included both null models as both are used in the literature and at the moment neither has been shown to be a better choice than the other.

2.3.3 Type I error rate estimations

Type I error rates were estimated for the randomly generated matrices by evaluating them using the appropriate null model and test statistic(s). As all matrices were generated randomly, species co-occurrences should also be random. Consequently, the null models should fail to find significant patterns of co-occurrence ($P < 0.05$). To test the effect of matrix dimension on type I error rates we generated 10,000 matrices for each possible combination of *m* species, *m* = {3, 4, 5,... , 17, 18, 19, 20, 25, 30, 35, 50}, by *n* plots, *n* = {3, 4, 5,... , 12, 13, 14, 15, 20, 25, 30, 35, 50, 75, 100, 150} for a total of 4,620,000 matrices. Species trait values were generated for each presence-absence matrix to test the error rates of the limiting similarity null models. Trait values were generated by drawing values from a uniform distribution constrained to the set of numbers $\{x \in \mathbb{R} \mid 0 < x \leq 100\}$ and limited to two decimal places by truncation. Truncation was done to represent a realistic level of precision and matched the maximum precision possible based on the dimension of the generated communities and test statistics used.

For each matrix we calculated "observed" values for each test statistic (C-Score, NN and SDNN). A distribution of "expected" values was then generated by running the null models for each matrix and the cumulative frequency of expected observations <, =, and > the observed value were determined. Three methods were used to assess significance. The first, the norm for these types of analyses and the method used by the EcoSim software (Gotelli & Entsminger, 2001), assesses significance based on the number of expected values that are more extreme or equal to the observed value that is, ≤ or ≥ the observed value. We refer to this method as the "inclusive" method. The second, "exclusive", method assesses significance based on expected values that are more extreme (< or >) than the observed value. The exclusive approach is consistent with normal hypothesis testing, which fails to reject the null when there is equality in the observed versus expected results (see Skipper, Guenther, & Nass, 1967 for a discussion of statistical significance). The third method, based on the Standardized Effect Size (SES) (Gotelli & Rohde, 2002; see Gurevitch, Morrow, Wallace, & Walsh, 1992), was included as it is both commonly used and available in the output of EcoSim. In this case |SES| > 1.96 were considered significant. An SES value of ±1.96 is equivalent to a 95% confidence interval. SES was calculated as $(Obs - Mean)/Std$,

where *Obs* is the observed value for each test statistic, and *Mean* and *Std* are the mean and standard deviation respectively of all randomizations for each test statistic of the null models. It should be noted that for small matrices it was not uncommon to have a standard deviation that was either equal to zero or approaching zero, which resulted in extreme SES values ($|SES| \geq 20$). The cause of these extreme values were matrices that, when shuffled, resulted in extremely small or zero difference in the values for the test statistic. Extreme values occurred with both the AWTS and the AITS null models but were more prevalent with AWTS and the SDNN metric (proportion of outliers by method; AWTS-SDNN: 0.0592, AWTS-NN: 0.0010, AITS-SDNN: 0.0368, AITS-NN: 0.0005). These outlier values were removed from the comparison of SES values but were retained for all other analyses (see Figure C-1).

2.3.4 Type II error rate estimations

Type II error rates represent the probability that real patterns of co-occurrence or limiting similarity will go undetected by the null models. A similar approach to the type I error rate estimations was used, however for these tests the amount of structure (pattern) in each matrix was increased from the initial random state and the resulting matrices tested with the null models. To do this both presence-absence matrices and species trait values were randomly generated using the methods described above for type I error rates. The number of species and plots used for the co-occurrence tests were *m* species, *m* = {5, 10, 15, 20, 25, 30, 35}, by *n* plots, *n* = {3, 4, 5, …, 13, 14, 15, 20, 25, 30, 35, 50}. The number of species and plots used for the limiting similarity tests were *m* species, *m* = {5, 10, 15, 20, 25, 30, 35}, by *n* plots, *n* = {3, 4, 5, …, 13, 14, 15, 20, 25, 30, 35, 50, 75, 100, 150, 200}. For both co-occurrence and limiting similarity tests 10,000 presence-absence matrices were generated for each combination of $m \times n$. The co-occurrence null model used a reduced set of plots compared to the limiting similarity null models due to the substantially longer processing time required for the co-occurrence null model tests.

Analyzing type II errors required the introduction of non-random patterns in the presence-absence matrices for the co-occurrence null model tests. To produce matrices with maximal C-Scores, the initial or observed C-Score was first determined after which each species was re-assigned among the plots by using a Fisher-Yates shuffle (Fisher,

1963). Once all species had been shuffled among plots the C-Score was re-calculated. If the new C-Score was greater than the previous value then the location of the species presences with respect to plots was stored. This shuffling was done 10,000 times in an attempt to determine a maximal C-Score value for each matrix. This approach ensured that each matrix had more extreme negative co-occurrence than had initially been introduced by random chance.

For the limiting similarity null models a similar process was used to maximize trait structure. One of the expectations under limiting similarity is that species avoid competition through trait differences; we remained consistent with this expectation by only adding trait structure to matrices with no negative co-occurrence. A pairwise test of species co-occurrence was used (Veech, 2013) to select matrices with no negative co-occurrence. Veech's (2013) method of testing for co-occurrence was used due to it's ability to rapidly test for co-occurrence between species pairs and not just at the community level that is, it ensured that there was no negative co-occurrence in the matrices. Trait value assignment for these matrices was done iteratively by selecting trait values at random and assigning them sequentially to species. With each assignment the values of NN and SDNN were calculated and stored. This was repeated 100,000 times for each matrix, re-assigning trait values from the pool at random with each repetition. With each re-assignment, if the new values for the NN and SDNN were greater than the stored values, the species and trait combinations were stored, resulting in maximal values for the test metrics. Once these maximal values were determined the limiting similarity null models were run for the matrix, which contained introduced, non-random trait structure.

As with the type I error rate estimations, statistical significance was determined by calculating the proportion of null models that resulted in more extreme values than the observed value. In the case of the type II estimations, however, our expectation was that our non-random matrices would have more structure than expected by chance (since we increased structure in all cases) and as a result tests were one tailed.

2.3.5 Software

All code for the null model analyses were written in Scala (Version 2.9.2) [Computer Language], available from http://www.scala-lang.org/downloads using IntelliJ IDEA Community Edition (Version 12.1.3) [Computer program], retrieved from http://www.jetbrains.com/idea/download/index.html and run on the Java VM (Version 1.6) [Computer software], available from http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html. Statistical analyses (non null model) were carried out using the R Project for Statistical Computing (Version 3.0.3) [Computer software], available at http://cran.r-project.org. All code used for this project is available in Appendix A and Appendix B as well as online at https://github.com/lavendermi.

2.4 Results

No significant relationships were found between co-occurrence null model SES values and Shannon's E (r = -0.0132, p = 0.6769), Simpson's E (r = --0.0007, p = 0.9829) or slope of the log-transformed Rank Abundance curve (r = 0.0275, p = 0.3853) (Figure 2-1). For each presence-absence matrix we also compared the SES values from the associated trait null models and the co-occurrence null model to ensure that the limiting similarity tests were independent of co-occurrence (Figure C-1).

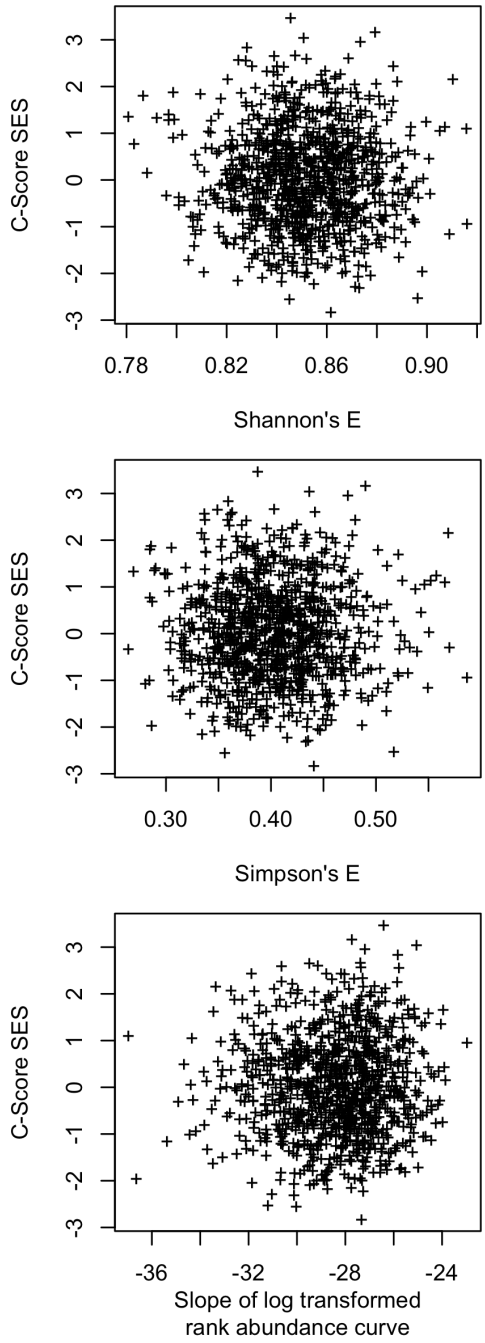

Figure 2-1. Plots represent three different measures of species evenness with respect to Standardized Effect Size (SES) of the co-occurrence null model. Data are from 1000 randomly generated 50 × 50 matrices and each data point represents 1000 null models for each matrix.

2.4.1 Type I Error Rate Estimation

Type I error rates for all three null models were sensitive to both the dimension of the matrices and the method of determining significance (Figure 2-2 and Figure 2-3). Type I error rates of the co-occurrence null models increased with both decreasing species numbers and decreasing plot numbers however, the number of species had a greater impact. The use of an inclusive *p*-value for determining significance resulted in almost all null model tests exceeding an α value of 0.05 (Figure 2-2); however, for the majority of test matrices containing greater than 6 species an $\alpha < 0.10$ was obtained. Exclusive *p*-values for significance determination resulted in type I error rates of less than $\alpha = 0.10$ (Figure 2-2) with the majority of tests falling in the range of $\alpha < 0.05$. This was contrary to that of inclusive *p*-values, where increasing matrix dimension resulted in slightly increasing error rates.

Using SES to determine significance resulted in type I error rates exceeding $\alpha = 0.05$ for test matrices containing fewer than five species (Figure 2-2). For matrices with three species the error rate increased with increasing plot number. For matrices with greater than five species, however, the use of SES for significance determination resulted in error rates with an $\alpha <= 0.05$.

AITS in combination with the NN trait metric resulted in error rates below 0.05 in all cases when an exclusive *p*-value was used (Figure 2-3). Error rates exceeded 0.05 when inclusive *p*-values were used in combination with six species or less. The use of SES with three × three matrices also resulted in error rates higher than 0.05. AITS in combination with SDNN had similar results, the exception being that error rates exceeded 0.05 for both the Inclusive and SES measures when species numbers were less than seven and six respectively. Inclusive and SES also had a slight upward trend as the number of plots increased.

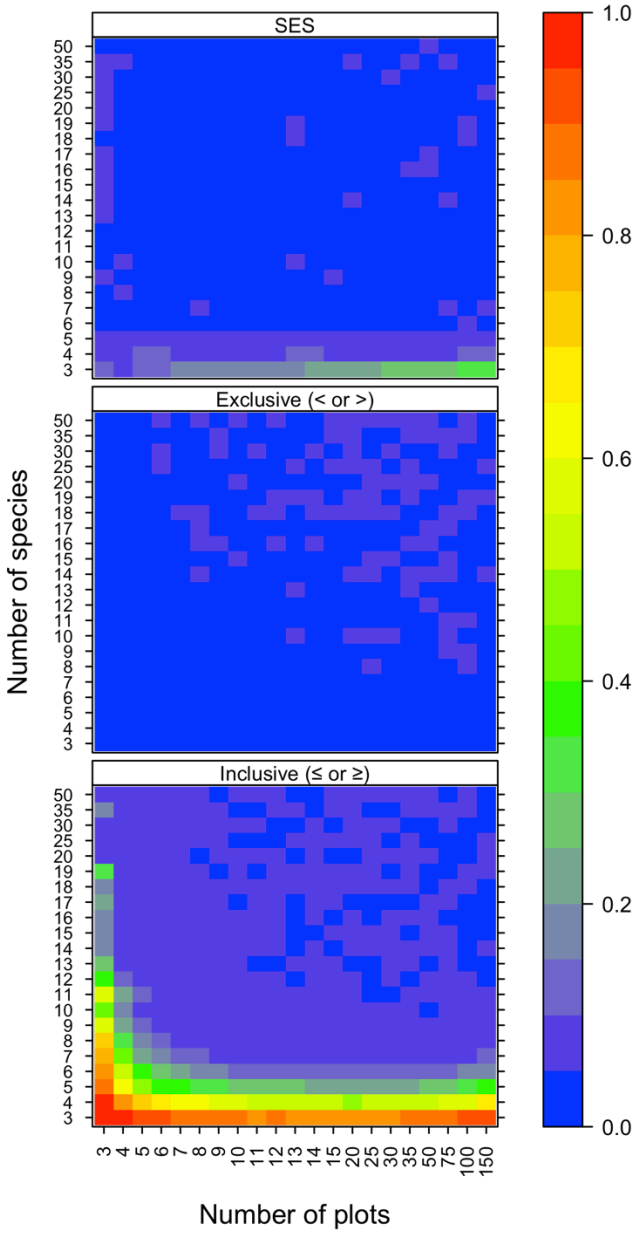

Figure 2-2. Type I error rates of the co-occurrence null model test. Each panel represents a different criterion for determining the significance of the null model; a) inclusive *p*-values (<= or >=), b) exclusive *p*-values (< or >), and c) SES. The colour of each cell indicates the proportion of the 10,000 null models that were significant for that combination of species by plots. Blue cells indicate lower type I error rates and red cells indicate higher type I error rates.

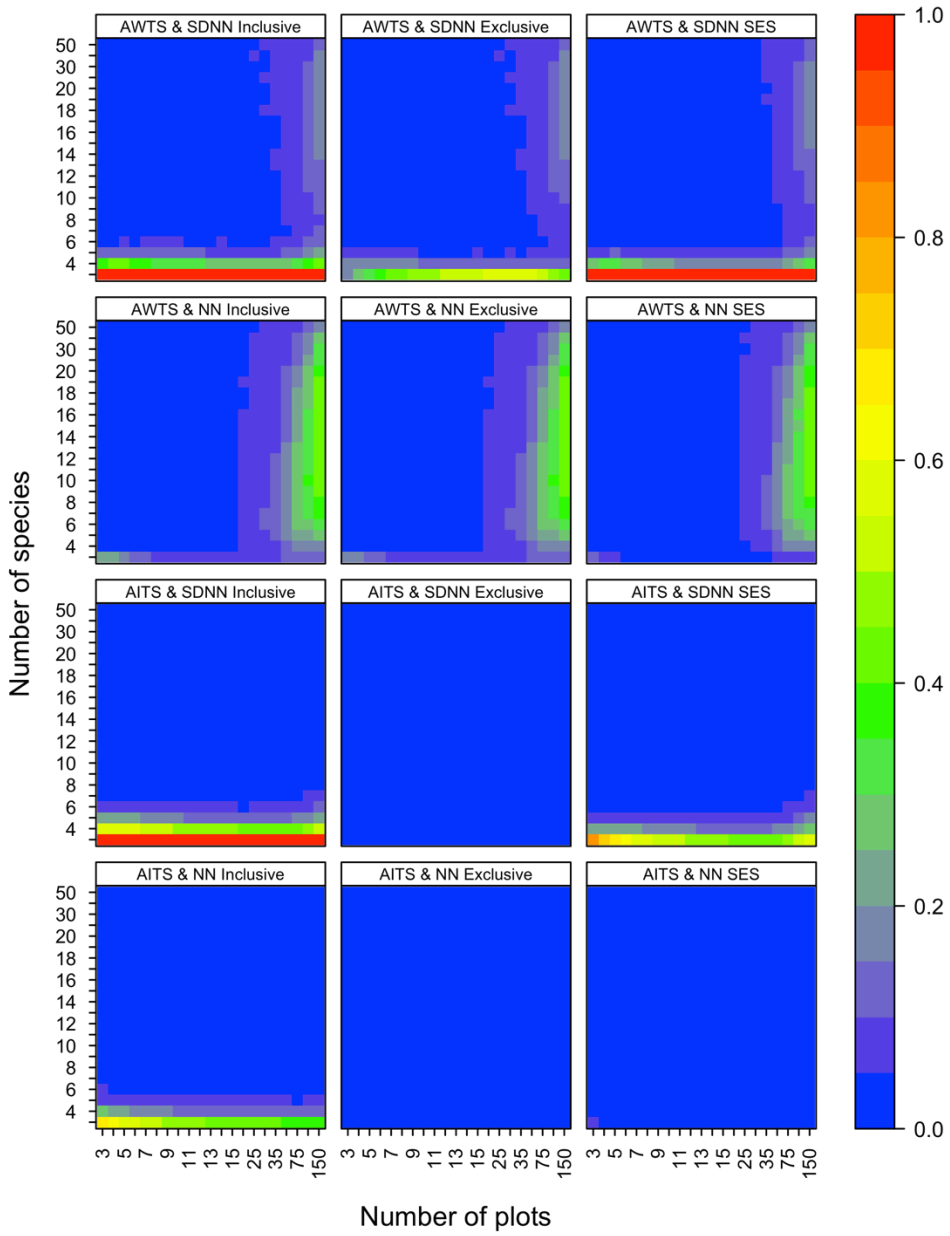

Figure 2-3. Type I error rates of the limiting similarity null models. Each panel represents a different combination of randomization algorithm, metric and criterion for determining the significance of the null model. The colour of each cell indicates the proportion of the 10,000 null models that were significant for that combination of species by plots. Blue cells indicate lower type I error rates and red cells indicate higher type I error rates.

2.4.2 Type II Error Rate Estimation

The negative co-occurrence null model with an inclusive significance criterion was able to detect negative co-occurrence patterns in at least 95% of the test matrices when there were fewer than thirteen plots however, this rate decreased to 50% as the number of plots increased to fifty (Figure 2-4). The number of species in the matrices had only a marginal effect on the type II error rates, regardless of significance criterion, with increasing species number resulting in a slight decrease in type II error rates.

The exclusive and SES significance criterion showed similar results to the inclusive criterion with three exceptions. Type II error rates exceeded 0.05 in cases where species numbers were less than ten, in matrices with three plots and less than fifteen species using the exclusive significance criterion, and in matrices with three plots and ten or fewer species using the SES significance criterion. In these cases the error rates ranged between 10% and 75%. AITS in combination with the SDNN metric produced more type II errors as the number of plots increased; however, increasing the number of species seemed to counteract the affect that the increasing number of plots had.

Type II error rates for both limiting similarity null models were generally very stable with $\alpha <= 0.05$ for all matrices and significance criteria with more than five species (Figure 2-5). The AITS null model with both the NN metric resulted in error rates <= 0.05 for all matrix dimensions. With fewer than five species AWTS error rates decreased as the number of plots increased.

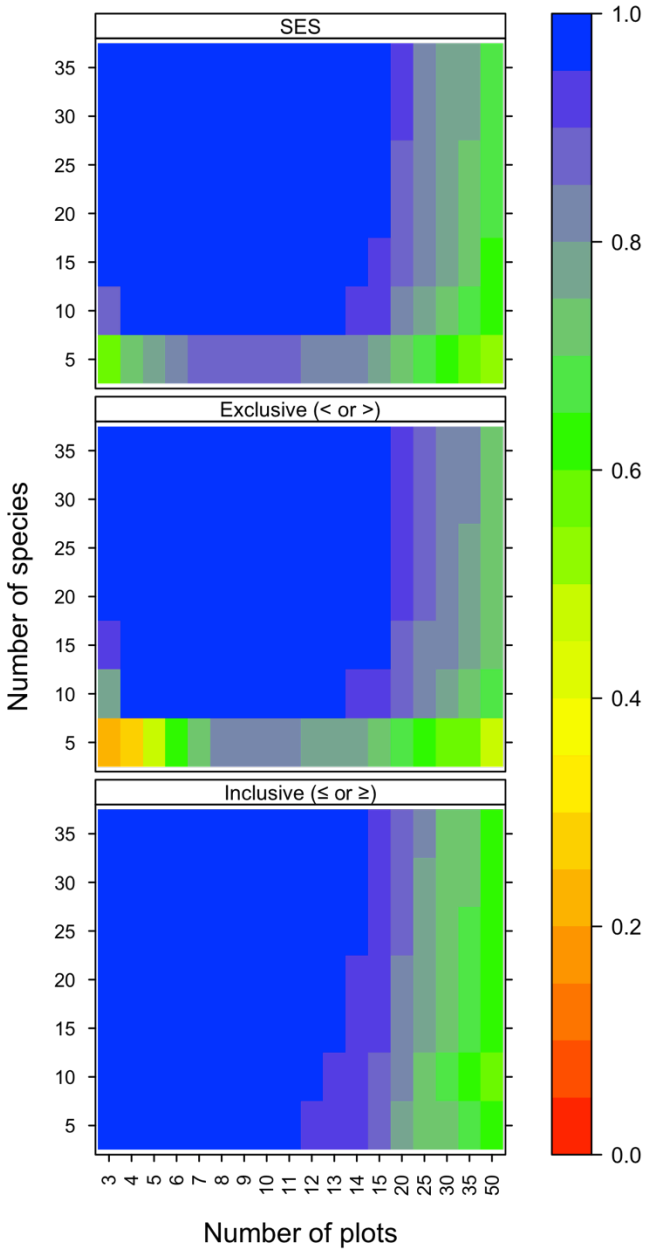

Figure 2-4. Type II error rates of the co-occurrence null model test. Each panel represents a different criterion for determining the significance of the null model. The colour of each cell indicates the proportion of the 10,000 null models that were significant for that combination of species by plots. Blue cells indicate lower type II error rates and red cells indicate higher type II error rates.

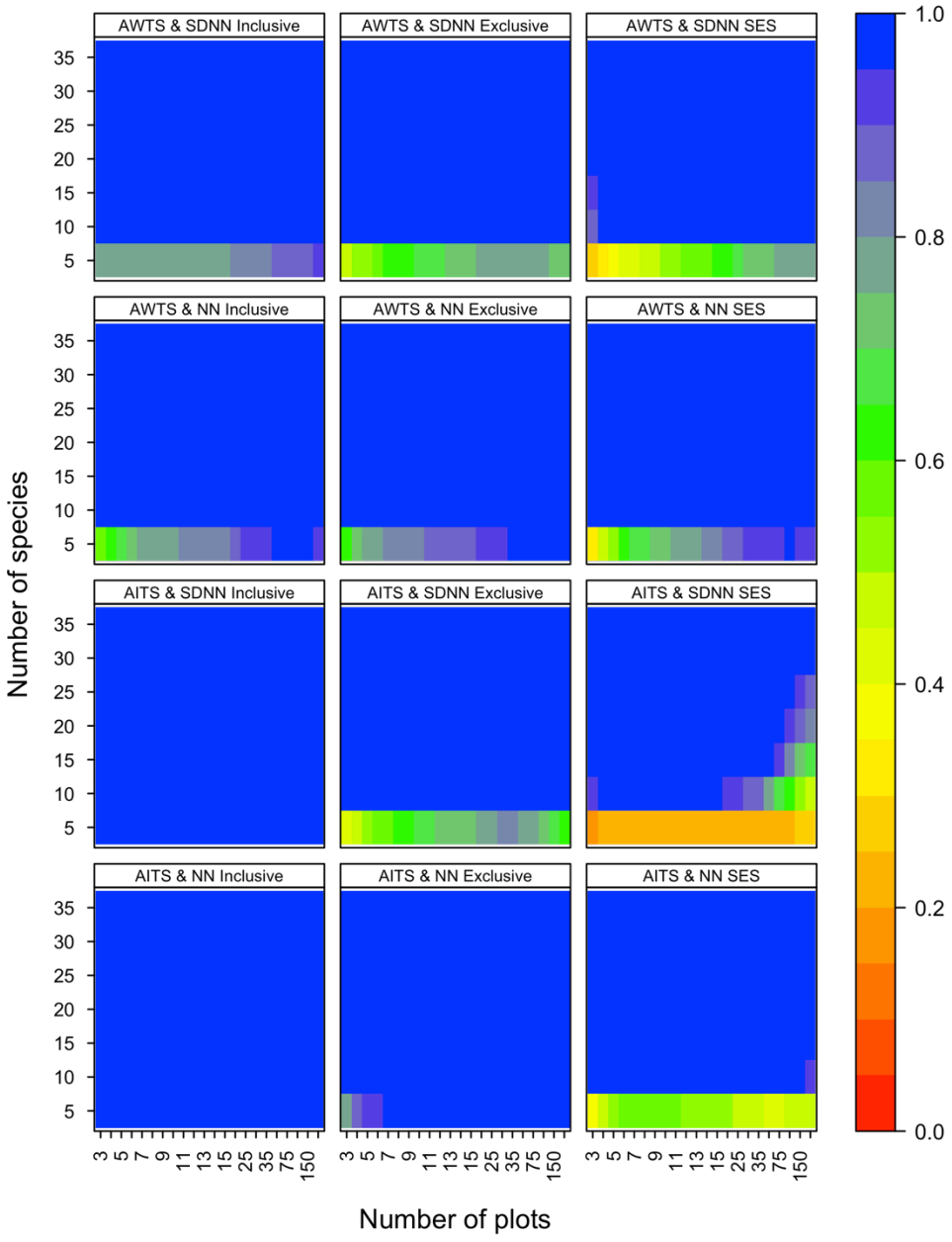

Figure 2-5. Type II error rates of the limiting similarity null models. Each panel represents a different combination of randomization algorithm, metric and criterion for determining the significance of the null model. The colour of each cell indicates the proportion of the 10,000 null models that were significant for that combination of species by plots. Blue cells indicate lower type II error rates and red cells indicate higher type II error rates.

2.5 Discussion

2.5.1 Type I Error Rate Estimation

2.5.1.1 Negative co-occurrence

Our results indicate that negative co-occurrence null models become susceptible to type I errors as the number of species decreases; with the sensitivity dependent on the method used to assess significance. The usual method of determining significance by using an inclusive P–value not only resulted in unacceptable error rates for matrices with smaller numbers of species (< 8), but also resulted in an overall error rate of approximately 10% for matrices with larger numbers of species (8 to 50 in this study). While the error rate of 10% is consistent with the findings of Gotelli (2000), it can be improved upon by using an exclusive *p*-value (< or >) for significance determination. The use of an exclusive *p*-value resulted in no type I error rates exceeding 10% for any of the matrix dimensions considered and reduced the majority of the null model error rates to a more desirable five percent or less. One reason that the use of an inclusive *p*-value may lead to increased error rates, particularly for small matrices, is that each matrix can only be shuffled a limited number of ways. Imposing restrictions, such as maintaining row and column totals, further restricts the number of possible shuffles (Figure 2-6). This means that, for smaller matrices in particular, the likelihood of the randomization producing the original "observed" matrix increases resulting in comparisons of the initial matrix to the set of null matrices of which it is a member.

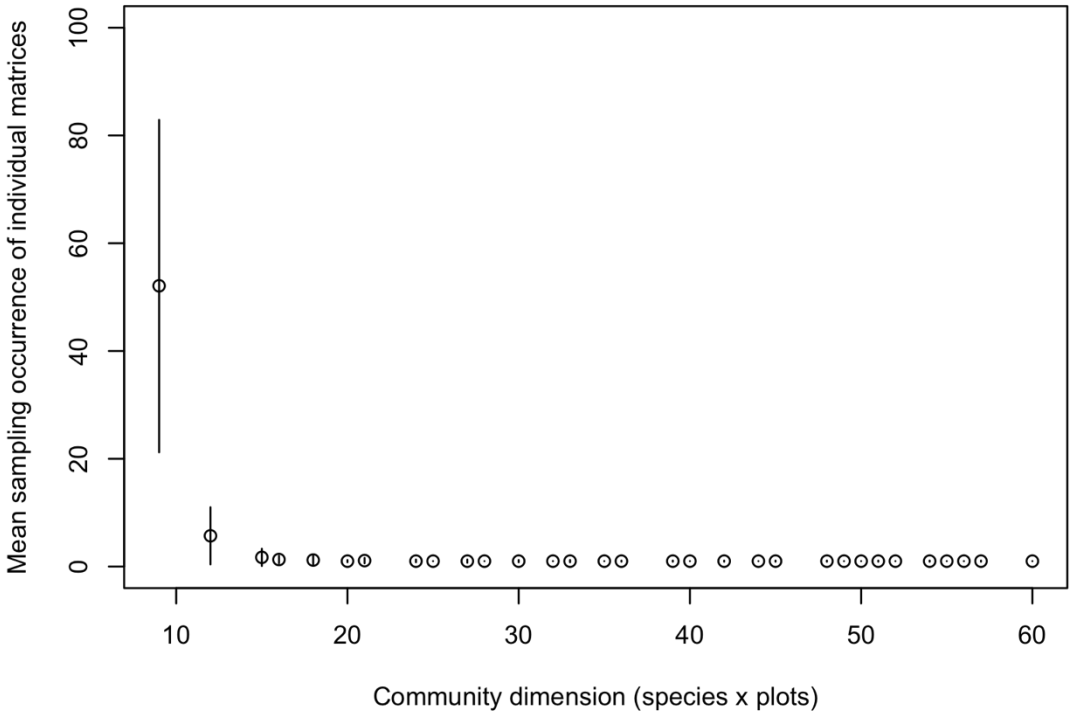

Figure 2-6. Plot showing the number of times specific matrix configurations are generated with respect to matrix dimension. Circles are the mean counts and lines represent 1 standard deviation. Data is based on the generation of 10,000 matrices for each combination of plot by species (see the presence-absence section of the methods for details).

Using SES to determine significance is becoming commonplace (Gotelli & McCabe, 2002; e.g., Gotelli & Rohde, 2002; Heino, 2013; Ulrich & Gotelli, 2007) however, our results suggest that an exclusive *p*-value approach is better for type I error rates when matrices with fewer species are tested. SES does provide a convenient way to compare the results between matrices however, it should be noted that SES (or *z*-score) assumes that the underlying distribution is normal and deviation from normality may be problematic.

2.5.1.2 Limiting similarity - AITS

The AITS randomization algorithm performed well for both the NN and SDNN metrics when the inclusive significance criterion was used and the number of species was greater than five for NN and greater than six for the SDNN metric regardless of the number of plots. Like the negative co-occurrence null model however, error rates for matrices with fewer species (< 6) fell below five percent when an exclusive significance criterion was used. As with the negative co-occurrence models, this likely results from the generation of random matrices that match the observed matrix being tested. Using SES to determine significance showed a similar pattern to that of co-occurrence. Specifically, SES improved the type I error rates for the AITS-NN combination compared to the inclusive measure but only had a moderate improvement for the AITS-SDNN combination. In general our results were consistent with those of Hardy (2008). Hardy's "1a" null model, the equivalent to our AITS null model, was also the best model for type I error rates overall. It should be noted however, that Hardy used inclusive *p*-values to determine significance. It is likely that he would have improved type I error rates if he had also used an exclusive significance criterion.

2.5.1.3 Limiting similarity – AWTS

We found limiting similarity null models that used the AWTS randomization algorithm and NN metric had excellent type I error rates for matrices with as few as four species. There was, however, a significant increase in error rates with increasing plot number. This trend was consistent for all three significance criteria with 30 to 35 plots resulting in error rates exceeding 10% when species numbers were low. Error rates rose to 10% or higher for all matrices when the number of plots exceeded 35.

AWTS in combination with the SDNN metric resulted in error rate patterns that resembled the combined patterns of the AITS-NN and AWTS-NN error rates. Type I error rates for matrices with greater than five species were less than five percent, however increasing the number of plots resulted in increasing error rates with 35 or more plots resulting in errors exceeding 10%. These results are consistent with Hardy's (2008) as well. In this case Hardy's "1s" null model, functionally equivalent to our AWTS algorithm, also resulted in higher type I error rates in general.

2.5.2 Type II Error Rate Estimation

2.5.2.1 Negative co-occurrence

Our results indicate that type II error rates for the negative co-occurrence null model are acceptable for matrices with small plot numbers (< 15). This is contradictory to the work of Gotelli (2000; 2011) and Fayle and Manica (2010) which indicates that increasing matrix sizes are more likely to result in type I errors. This increase in type I errors should result in an associated decrease in type II errors, which we did not find in our initial analyses. We suspect that the increase in type II errors with increasing matrix size and, in particular, with increasing plot number may have been an artefact of the method used to add structure to the matrices. The artefact may have resulted from insufficiently sampling the possible combinations of species re-arrangements as we maximized the C-Score for the type II error rate analyses. In order to investigate this as the cause of the increasing error rates we ran a reduced set of type II negative co-occurrence null model tests for a set of matrices with 35 plots increasing the number of shuffles used to maximize the C-Score from 10,000 to 1,000,000. It was evident from these results (Figure C-2) that the increasing type II error rates of the null models with increasing plot number was attributed to insufficiently sub-sampling from the pool of possible arrangements. These results combined with the results of the full set of type II error rates for the co-occurrence null model indicate that the co-occurrence null model is robust to type II errors across all community dimensions.

2.5.2.2 Limiting similarity-AITS

The ability of the limiting similarity null model using the AITS randomization algorithm to detect patterns of non-random trait distribution was very high (>= 95%) when the inclusive *p*-value was used. It's statistical power was only slightly reduced when exclusive *p*-values or SES were used. These patterns of type II error rates are consistent with type I error rates in that an increase in type I errors had an associated decrease in type II errors. As with the type I error rates, the AITS null model was sensitive to the significance criteria and the number of species with fewer species resulting in increased type I errors. Unlike the type I error rate estimations however, error rates did not improve when exclusive *p*-value or SES were used. Once again, this is consistent with the type I errors as there is always a trade off between type I and type II error rates.

2.5.2.3 Limiting similarity-AWTS

The AWTS null model also had good type II error rates for all numbers of plots however, as with the AITS null model, type II errors for matrices with fewer than ten species were high. The AWTS null model did not show the same pattern of increasing error rate with the exclusive *p*-value criterion and SES as the co-occurrence and other limiting similarity null models did. These results are, once again, consistent with Hardy's (2008) work for matrices of the same size but do extend our knowledge for matrices of smaller dimension.

2.6 Conclusion and Recommendations

Our study was carried out to extend the works of both Gotelli (2000) and Hardy (2008) by determining if there are any constraints on the use of either limiting similarity or co-occurrence null models. We found that there exists a lower threshold of species number (6) common to all of the null models tested with no common upper limit. The only indication of an upper limit existed for the abundance-weighted trait shuffling algorithm (AWTS) which showed an increase in type I error rates when the number of plots exceeded 25. It is evident from these results that the utility of the AWTS null model is limited and should be avoided except under very specific conditions. It is also evident from our work that the abundance-independent trait shuffling algorithm (AITS) in combination with either the

mean nearest neighbour distances (NN) or standard deviation of the nearest neighbour distances (SDNN) is a reliable tool for assessing limiting similarity within communities.

In summary, we recommend the use of exclusive *p*-values for both the limiting similarity and co-occurrence null models. Exclusive *p*-values reduce the type I error rates without a significant increase in type II error rates. The use of C-Score and the SIM9 (fixed-fixed) null model is robust for the analysis of co-occurrence patterns even with smaller dimensioned matrices; however, its use should be restricted to matrices with five or more species to avoid the possibility of type II errors. We also recommend the use of the AITS null model over AWTS with either the NN or SDNN metric. The one caveat for the limiting similarity null model is that the use of NN results in lower type II error rates for smaller matrices than SDNN.

2.7 References

Baraloto, C., Hardy, O. J., Paine, C. E., Dexter, K. G., Cruaud, C., Dunning, L. T., et al. (2012). Using functional traits and phylogenetic trees to examine the assembly of tropical tree communities. Journal of Ecology, 100, 690–701.

Bell, G. (2000). The Distribution of Abundance in Neutral Communities. The American naturalist, 155, 606–617.

Burns, K. C. (2007). Patterns in the assembly of an island plant community. Journal of biogeography, 34, 760–768.

Connor, E. F., & Simberloff, D. (1979). The assembly of species communities: Chance or competition? Ecology, 60, 1132–1140.

Cornwell, W. K., & Ackerly, D. D. (2013). Community assembly and shifts in plant trait distributions across an environmental gradient in coastal California, 79, 109–126.

Dante, S. K., Schamp, B. S., & Aarssen, L. W. (2013). Evidence of deterministic assembly according to flowering time in an old-field plant community, 27, 555–564.

de Bello, F., Thuiller, W., Lepš, J., Choler, P., Clément, J.-C., Macek, P., et al. (2009). Partitioning of functional diversity reveals the scale and extent of trait convergence and divergence. Journal of Vegetation Science, 20, 475–486.

Diamond, J. M. (1975). Assembly of species communties. In Ecology and evolution of communities (pp. 342–444). Cambridge, MA, US.: Belknap Press of Harvard University Press.

Fayle, T. M., & Manica, A. (2010). Reducing over-reporting of deterministic co-occurrence patterns in biotic communities. Ecological Modelling, 221, 2237–2242.

Fisher, R. A. (1963). Statistical Tables for biological, agricultural and medical research (6 ed.). London: Oliver and Boyd. Retrieved from http://www.worldcat.org/title/statistical-tables-for-biological-agricultural-and-medical-research/oclc/248830531?referer=di&ht=edition

Franzén, D. (2004). Plant species coexistence and dispersion of seed traits in a grassland. Ecography, 27, 218–224.

Gainsbury, A. M., & Colli, G. R. (2003). Lizard Assemblages from Natural Cerrado Enclaves in Southwestern Amazonia: The Role of Stochastic Extinctions and Isolation. Biotropica, 35, 503–519.

Gotelli, N. J. (2000). Null model analysis of species co-occurrence patterns. Ecology, 81, 2606–2621.

Gotelli, N. J., & Entsminger, G. L. (2001). EcoSim: Null models software for cology. Version 7. Acquired Intelligence Inc. & Kesey-Bear, Jericho, VT. Retrieved from http://homepages.together.net/~gentsmin/ecosim.htm

Gotelli, N. J., & McCabe, D. J. (2002). Species co-occurrence: A meta-analysis of J. M. Diamond's assembly rules model. Ecology, 83, 2091–2096.

Gotelli, N. J., & Rohde, K. (2002). Co-occurrence of ectoparasites of marine fishes: A null model analysis. Ecology Letters, 5, 86–94.

Gotelli, N. J., & Ulrich, W. (2011). Over-reporting bias in null model analysis: A response to Fayle and Manica (2010). Ecological Modelling, 222, 1337–1339.

Götzenberger, L., de Bello, F., Bråthen, K. A., Davison, J., Dubuis, A., Guisan, A., et al. (2012). Ecological assembly rules in plant communities--approaches, patterns and prospects. Biological reviews of the Cambridge Philosophical Society, 87, 111–127.

Gurevitch, J., Morrow, L. L., Wallace, A., & Walsh, J. S. (1992). A Meta-Analysis of Competition in Field Experiments. The American naturalist, 140, 539–572.

Hardy, O. J. (2008). Testing the spatial phylogenetic structure of local communities: Statistical performances of different null models and test statistics on a locally neutral community. Journal of Ecology, 96, 914–926.

Heino, J. (2013). Environmental heterogeneity, dispersal mode, and co-occurrence in stream macroinvertebrates. Ecology and evolution, 3, 344–355.

Kraft, N. J. B., & Ackerly, D. D. (2010). Functional trait and phylogenetic tests of community assembly across spatial scales in an Amazonian forest. Ecological Monographs, 80, 401–422.

Kraft, N. J. B., Valencia, R., & Ackerly, D. D. (2008). Functional traits and niche-based tree community assembly in an Amazonian forest. Science (New York, N.Y.), 322, 580–582.

Mayfield, M. M., Boni, M. F., Daily, G. C., & Ackerly, D. D. (2013). SPECIES AND FUNCTIONAL DIVERSITY OF NATIVE AND HUMAN-DOMINATED PLANT COMMUNITIES. Ecology, 86, 2365–2372.

Mouillot, D., Dumay, O., & Tomasini, J. A. (2007a). Limiting similarity, niche filtering and functional diversity in coastal lagoon fish communities. Estuarine, Coastal and Shelf Science, 71, 443–456.

Mouillot, D., Mason, N. W. H., & Wilson, J. B. (2007b). Is the abundance of species determined by their functional traits? A new method with a test using plant communities. Oecologia, 152, 729–737.

Mouillot, D., Stubbs, W. J., Faure, M., Dumay, O., Tomasini, J. A., Wilson, J. B., & Chi, T. D. (2005). Niche overlap estimates based on quantitative functional traits: a new family of non-parametric indices. Oecologia, 145, 345–353.

Preston, F. W. (1962a). The Canonical Distribution of Commonness and Rarity: Part I. Ecology, 43, 185–215.

Preston, F. W. (1962b). The Canonical Distribution of Commonness and Rarity: Part II. Ecology, 43, 410–432.

Schamp, B. S., & Aarssen, L. W. (2009). The assembly of forest communities according to maximum species height along resource and disturbance gradients. Oikos, 118, 564–572.

Schamp, B. S., Chau, J., & Aarssen, L. W. (2008). Dispersion of traits related to competitive ability in an old-field plant community. Journal of Ecology, 96, 204–212.

Schamp, B. S., Hettenbergerová, E., & Hájek, M. (2011). Testing community assembly predictions for nominal and continuous plant traits in species-rich grasslands. Preslia, 83, 329–346.

Schamp, B. S., Horsák, M., & Hájek, M. (2010). Deterministic assembly of land snail communities according to species size and diet. Journal of Animal Ecology. doi:10.1111/j.1365-2656.2010.01685.x

Skipper, J. K., Guenther, A. L., & Nass, G. (1967). The sacredness of. 05: A note concerning the uses of statistical levels of significance in social science. The American Sociologist, 2, 16–18.

Smith, B., Moore, S. H., & Grove, P. B. (1994). Vegetation texture as an approach to community structure: community-level convergence in a New Zealand temperate rainforest. New Zealand ….

Stone, L., & Roberts, A. (1990). The checkerboard score and species distributions. Oecologia, 85, 74–79.

Stubbs, W. J., & Wilson, J. B. (2004). Evidence for limiting similarity in a sand dune community. Journal of Ecology, 92, 557–567.

Ulrich, W., & Gotelli, N. J. (2007). Null model analysis of species nestedness patterns. Ecology, 88, 1824–1831.

Ulrich, W., & Gotelli, N. J. (2010). Null model analysis of species associations using abundance data. Ecology, 91, 3384–3397.

Ulrich, W., Ollik, M., & Ugland, K. I. (2010). A meta-analysis of species–abundance distributions - Ulrich - 2010 - Oikos - Wiley Online Library. Oikos.

Veech, J. A. (2013). A probabilistic model for analysing species co-occurrence. Global Ecology and Biogeography, 22, 252–260.

Weiher, E., Clarke, G. D. P., & Keddy, P. A. (1998). Community assembly rules, morphological dispersion, and the coexistence of plant species. Oikos, 81, 309–322.

Wilson, J. B. (1989). A Null Model of Guild Proportionality, Applied to Stratification of a New Zealand Temperate Rain Forest. Oecologia, 80, 263–267.

Wilson, J. B., & Stubbs, W. J. (2012). Evidence for assembly rules: Limiting similarity within a saltmarsh. Journal of Ecology, 100, 210–221.

Chapter 3.0 Preamble

Chapter 2 established that negative co-occurrence and limiting similarity null models have acceptable type I and type II error rates provided that the number of species in the matrix is six or greater, that the use of exclusive *p*-values for significance determination was more robust than inclusive *p*-values particularly as the number of species in the matrix decreased, and that abundance independent trait shuffling was a better choice than abundance weighted trait shuffling for limiting similarity analyses.

Chapter 3 builds on the results of the previous chapter by investigating the statistical power of the negative co-occurrence and limiting similarity null model tests with respect to the proportion of species contributing signal and matrix dimension. The results of this chapter indicate that statistical power decreased rapidly as the proportion of species contributing to signal of co-occurrence and limiting similarity decreased. Below proportions of 80% the null model tests were unlikely to detect any significant pattern with the co-occurrence null model having only slightly better statistical power at proportions of 80% or less.

This chapter relates to the overall thesis by addressing the second objective of determining the statistical power of the null models with respect to matrix dimension and proportion of species contributing signal to the patterns being investigated.

3.0 Power analysis of limiting similarity and negative co-occurrence null models

3.1 Abstract

Limiting similarity and negative co-occurrence null models are commonly used tools for assessing the role of competition in community assembly. Negative co-occurrence is used to detect patterns resulting from competitive exclusion and limiting similarity to detect patterns of trait similarity/dissimilarity presumably associated with niche overlap/differentiation. While studies aimed at detecting significant patterns of negative co-occurrence have had moderate success, those aimed at detecting patterns of limiting similarity have had little success. Previous work investigating the statistical performance of these null models, in particular their type II error rates, have used matrices that may not be biologically realistic or have only considered patterns when all species contribute signal to the pattern being sought. The relatively low support for limiting similarity may be the result of the limiting similarity null model having reduced statistical power when only a proportion of species contribute signal to patterns of limiting similarity and negative co-occurrence. To explore the statistical performance of these models with respect to the proportion of species involved in limiting similarity and negative co-occurrence, we randomly generated presence-absence matrices covering a broad range of community dimensions. We then added patterns of limiting similarity and negative co-occurrence to matrices in known proportions and the null models were run for each matrix. The C-Score test statistic and "fixed-fixed" independent swap algorithm were used for co-occurrence null models. For limiting similarity null models, we used the mean Nearest Neighbour (NN) trait distance and the Standard Deviation of Nearest Neighbour trait distances (SDNN) as test statistics, and considered two randomization algorithms: Abundance Independent Trait Shuffling (AITS) and, Abundance Weighted Trait Shuffling (AWTS). Type II error rates for both null models increased rapidly with decreases in the proportion of species contributing to co-occurrence or limiting similarity patterns. Acceptable type II error rates ($\beta < 0.30$) were only achieved when 80% or more of species contributed signal to patterns of limiting similarity and when 60% or more of species contributed signal to patterns of negative co-occurrence. Limiting similarity and negative co-occurrence null models significantly under-report patterns that they are designed to detect, particularly when the pattern of interest is driven by a subset of species in the community. Further work to

improve the performance of these null models and / or alternative approaches to improve the detection of these patterns is required.

3.2 Introduction

Negative co-occurrence and limiting similarity are two types of null model commonly employed to assess the role of competition in community assembly. While both negative co-occurrence and limiting similarity attempt to detect patterns of species associations within a community, the approach each uses is slightly different. Negative co-occurrence attempts to detect patterns of species segregation resulting from competitive exclusion (Gause 1932) and is often associated with species that overlap in niche (Grime 2006, Mayfield and Levine 2010). Limiting similarity, on the other hand, attempts to detect species association patterns through the comparison of trait values. The expectation under limiting similarity is that species with similar traits are more likely to negatively co-occur and those with dissimilar traits are more likely to neutrally or positively co-occur. While null models have been used to look for patterns of both negative co-occurrence (e.g., Weiher et al. 1998, Gotelli and Rohde 2002, Gotelli and Ellison 2002, Ribichich 2005, Maestre and Reynolds 2007, Rooney 2008, Zhang et al. 2009, Maltez-Mouro et al. 2010) and limiting similarity (e.g., Weiher et al. 1998, Stubbs and Wilson 2004, Cornwell et al. 2006, Mouillot et al. 2007, Hardy 2008, Schamp et al. 2008, Kraft et al. 2008, de Bello et al. 2009, Schamp and Aarssen 2009, Kraft and Ackerly 2010, Wilson and Stubbs 2012), the amount of support for each varies. In a meta-analysis of community assembly processes Götzenberger *et al.* (2012) found that only 18% of studies discovered significant support for limiting similarity whereas, 41% of studies found co-occurrence to be a significant factor. Why is so little support found for limiting similarity when it is thought that it must exist (Stubbs and Wilson 2004).

One possible reason for the lack of support is that the null models may be asking a different question. That is, they are not asking if patterns of limiting similarity and negative co-occurrence exist but instead are asking if the community in question is shaped primarily by co-occurrence or limiting similarity. If only subsets of species within communities are involved in limiting similarity and negative co-occurrence, then the null models may simply miss these patterns, as they are not predominant but instead act in concert at the sub-

community level to shape whole communities. If this is the case, that we are failing to detect pattern at the sub-community level, then at what level of species involvement do the null models stop detecting pattern?

The statistical performance of both the negative co-occurrence and limiting similarity null models have been studied suggesting that they perform with acceptable type I and II error rates under many conditions (Gotelli 2000, Hardy 2008, Chapter Two, Fayle and Manica 2010). However, for the co-occurrence null models, neither Gotelli (2000) nor Fayle and Manica (2010) used biologically realistic matrices for their type II error rate estimations and Lavender et al. (Chapter Two) only investigated type II error rates for matrices in which all species contributed signal to patterns of either limiting similarity or negative co-occurrence. In the case of the limiting similarity statistical performance analysis, Hardy (2008) did not focus on the type II error rates of the null models. This resulted in insufficient detail on the methods used and in the results given to be able to make a fair assessment of type II error rates for the null models investigated.

One potential problem with the matrices used by Gotelli (2000) to examine the statistical power (type II error rates) of the co-occurrence null model is that the constraints used to create the matrices differed from those of their type I error rate estimations. That is, they did not have the same species richness and abundance constraints as the type I matrices. While the matrices used for the type I estimations were derived from a real data set, the West Indian finch data of Gotelli and Abele (1982), the type II matrix was derived from the theoretical matrix of Diamond and Gilpen (1982). The matrix used for type II error estimations was representative of a coarse grained distribution of species among extremely different habitat types (i.e. aquatic versus terrestrial), very different from the fine-grained distribution expected along the more subtle environmental gradients that characterize terrestrial communities. In more homogeneous communities, patterns of negative co-occurrence are noisy (i.e., imperfect) with pairs of species negatively co-occurring most, but not 100%, of the time. In Gotelli's (2000) analysis, the perfect checkerboard matrix used as the starting point is highly unlikely to occur in a natural community. While it is certainly important that a null model is able to identify strong

patterns such as this, understanding error rates among more realistic matrices is also of value.

In an evaluation of the sequential swap algorithm in combination with the C-Score metric of co-occurrence, Fayle and Manica (2010) tested matrices similar to those of Gotelli (2000) but also did not maintain species abundance and richness constraints across both type I and II error rate estimations. This resulted in matrices with different biological, and not necessarily realistic, constraints for the type II versus the type I error rate estimations. Lavender et al. (Chapter Two), on the other hand, maintained the same constraints for both the type I and II error estimations but only considered the case where all species in the community contributed signal. As a result, it remains uncertain how the negative co-occurrence null model performs, in terms of type II errors, as the proportion of species contributing signal to the pattern varies within a community.

While aspects of the type II error rates for the co-occurrence null model have been assessed (Gotelli 2000), no similar attempt has been made for two commonly used limiting similarity null models: Abundance Independent Trait Shuffling (AITS) or, Abundance Weighted Trait Shuffling (AWTS) (but see Hardy 2008). Type II error rates for these null models remain uncertain, particularly when different proportions of species contribute signal to the pattern under investigation. The purpose of this study was to estimate the type II error rates for the negative co-occurrence and limiting similarity (AWTS and AITS) null models using biologically realistic matrices with both a range of proportions of species contributing signal to the pattern and a broad range of matrix dimensions.

3.3 Methods

We tested two classes of null model, limiting similarity and negative co-occurrence. For the limiting similarity null models we tested two metrics, the mean nearest neighbour distances (NN) and the standard deviation of the nearest neighbour distances (SDNN) with two randomization schemes. The randomization schemes used were Abundance Independent Trait Shuffling (AITS) and Abundance Weighted Trait Shuffling (AWTS) (Dante et al. 2013). We used these schemes and metrics as both have been broadly used in the literature (e.g., Stubbs and Wilson 2004, Hardy 2008, Kraft et al. 2008, de Bello et al.

2009, Schamp and Aarssen 2009, Cornwell and Ackerly 2009, Wilson and Stubbs 2012, Dante et al. 2013) and the statistical performance of these particular combinations of metrics and randomization schemes is fairly well established in terms of type I error rates (Hardy 2008, Chapter Two). For the negative co-occurrence null model we tested the C-Score metric of Stone and Roberts (1990) in combination with the "fixed-fixed" independent swap algorithm (SIM9 in Gotelli 2000) randomization scheme as this combination also has well established type I error rates (Gotelli 2000, Chapter Two).

Non-degenerate matrices (Gotelli 2000) and trait values were generated using the same methods as in Chapter Two resulting in matrices with log-normal species abundances and uniformly distributed trait values. To estimate type II error rates when different proportions of species were contributing signal, we maximized the pattern of interest for varying proportions of species. Pattern was maximized for six different proportions of species in the test matrices; p = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0}. The matrices used for the co-occurrence tests consisted of all combinations of m species, m = {5, 10, 15, 20, 25, 30, 35}, by n plots, n = {3, 4, 5, …, 13, 14, 15, 20, 25, 30, 35, 50}. The matrices used for the limiting similarity tests consisted of all combinations of m species, m = {5, 10, 15, 20, 25, 30, 35}, by n plots, n = {3, 4, 5, …, 13, 14, 15, 20, 25, 30, 35, 50, 75, 100, 150, 200}. For both co-occurrence and limiting similarity tests 10,000 presence-absence matrices were generated for each combination of m species by n plots for a total of 16,800,000 matrices. Because we were interested in the relationship between co-occurrence and limiting similarity patterns, we focused our analysis of limiting similarity tests on presence-absence data, which are predominantly used for co-occurrence tests. The co-occurrence null model used a reduced set of plots compared to the limiting similarity null model due to the substantially longer processing time required for the randomization scheme of the co-occurrence null model tests. The same approach was used to add structure to the matrices as in Chapter Two; however, in this case structure was only added to the specified proportion of the species. To introduce non-random patterns in the presence-absence matrices for the co-occurrence null model tests, n species were selected at random where $n = m \times p$. To produce subsets of species with maximal C-Scores (i.e., high negative co-occurrence), the C-Score was first determined for the desired subset after which each species was re-assigned among the

plots using a Fisher-Yates shuffle (Fisher and Yates 1953). This method of shuffling the subset did result in matrices with different species richness for some plots as well as some plots containing no species. We chose not to adjust the matrix around the placement of the non-random species (those that we maximized the C-Score for) to maintain species richness/plot as that may have added unwanted structure elsewhere in the matrix. Once all species in the subset had been shuffled among plots the C-Score was re-calculated. If the new C-Score was greater than the previous value then the new matrix was stored. This process was repeated 10,000 times to maximize the C-Score for the subset under consideration. This approach ensured that each matrix had a subset of species with more extreme negative co-occurrence than had initially been introduced by random chance. This approach of assessing statistical power for the co-occurrence null model was very similar to that of Gotelli (2000) with three notable exceptions. First, we started with matrices that contained only noise and added structure while Gotelli (2000) started with extreme structure and added noise. Second, the amount of structure added was confined to a known proportion of species versus the random removal of structure across all species, and third, the underlying matrices were based on matrices with similar constraints to the type I error rate tests.

A similar process of adding pattern was used to maximize trait structure for the limiting similarity null models. Species were first sorted into two groups; those that negatively co-occurred with any other species and, those that did not. This grouping was done using a pairwise test of species co-occurrence (Veech 2013) with a *p*-value of 0.05. To add structure to the desired proportion of species we first determined the relative number of species (n) per matrix using the following equation, $n = m \times p$ were m is the number of rows in the matrix and p is the desired proportion. We randomly selected a subset of n species from the group of species not involved in negative co-occurrence (i.e. from the group of neutrally or positively co-occurring species). Traits were assigned to this subset of species by selecting trait values randomly from a uniform distribution and assigning them to species sequentially. After all species of the subset had been assigned a trait value, the values of NN and SDNN were calculated for the subset and stored. This was repeated 100,000 times, selecting n new species and trait values with each repetition. With each re-

assignment, if the new values for NN and SDNN were more extreme than the stored values, the new species and trait combinations were stored, resulting in maximal values for the test metrics. For NN the metric was maximized which was consistent with trait over-dispersion. For SDNN the metric was minimized which was also consistent with trait over-dispersion. Once these maximal values were determined for the subset of species, limiting similarity null models were run on the full matrix, which contained known proportions of introduced, non-random trait structure.

Type II error rates were determined by calculating the proportion of non-significant null models, or null models that did not result in values more extreme than the observed value. Because signal was always added by maximizing negative co-occurrence and patterns consistent with limiting similarity, all null model tests were one-tailed.

3.3.1 Software

All code for the null model analyses were written in Scala (Version 2.9.2) [Computer Language], available from http://www.scala-lang.org/downloads using IntelliJ IDEA Community Edition (Version 12.1.3) [Computer program], retrieved from http://www.jetbrains.com/idea/download/index.html and run on the Java VM (Version 1.6) [Computer software], available from http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html. Statistical analyses (non-null model) were carried out using the R Project for Statistical Computing (Version 3.0.3) [Computer software], available at http://cran.r-project.org. All code used for this project is available in Appendix A and Appendix E as well as online at https://github.com/lavendermi

3.4 Results

3.4.1 Co-occurrence null model

The ability of the negative co-occurrence null model to detect patterns of negative co-occurrence degraded as the proportion of species contributing signal decreased (Figure 3-1). With fewer than 60% of species contributing signal, the null model failed to detect significant pattern at $\alpha <= 0.30$ for all matrices with between nine and 15 species. Significant pattern at $\alpha <= 0.05$ was only detected when more than 80% of species

contributed signal and then only for more than 15 species and fewer than eight plots. The power of the co-occurrence null model increased slightly with increasing species number and decreasing plot number for proportions of contribution between 20% and 100% inclusive. When co-occurrence was not maximized for any species in the matrix (0% proportion), the probability of detecting significant negative co-occurrence was less than 0.10.

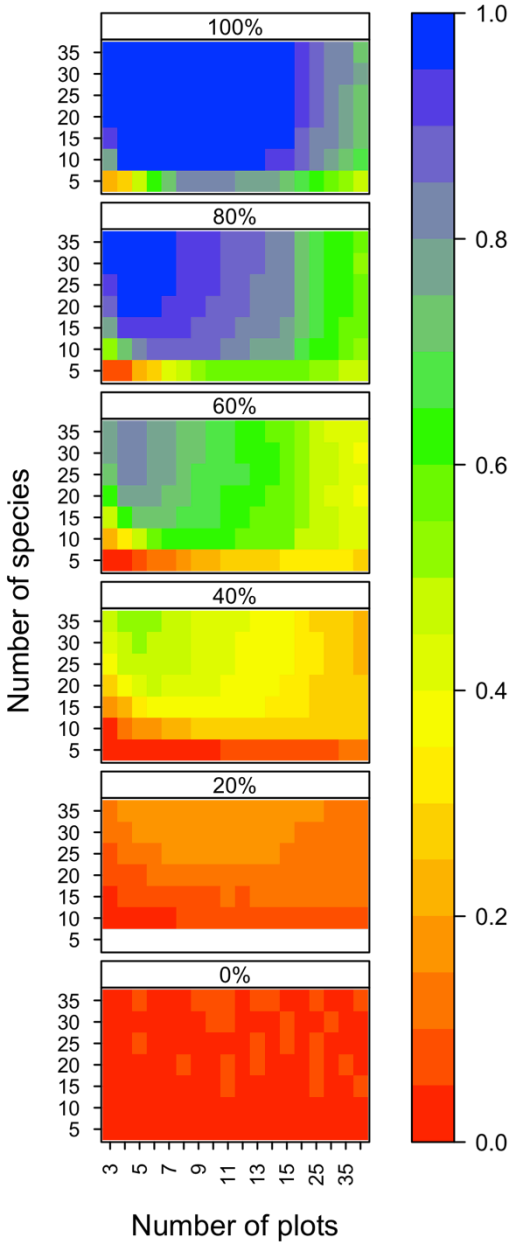

Figure 3-1. The power of the co-occurrence null model (1 – β) with respect to matrix dimension (number of species × number of plots) and the proportion of species contributing signal to the pattern of negative co-occurrence. Statistical power was estimated by maximizing the C-Score for set proportions of species prior to running the null model. The ability of the null model to detect the increased pattern of negative co-occurrence was determined using a *p*-value of 0.05 (one-tailed). Matrices were swapped 5000 times using the fixed-fixed independent swap algorithm of Gotelli (2000). Each cell within the plots represents the proportion of 10,000 null models that detected significant negative co-occurrence. The percentage of each plot represents the proportion of species

contributing signal to the pattern of negative co-occurrence. Blue shading indicates high statistical power (low type II error rates) and red shading indicates low statistical power (high type II error rates). White cells indicate no data and are associated with a proportions that result in single species for which it is impossible to calculate C-Score. The plot of 0% species proportion is equivalent to a type I error rate estimation and acted as a negative control for our method of analysis.

3.4.2 Limiting similarity null model

Type II error rates for the limiting similarity null model increased rapidly as the proportion of species contributing signal decreased (Figure 3-2); however, the error rates when using the standard deviation of the nearest neighbour distances (SDNN) metric increased at a faster rate than for the mean nearest neighbour distance (NN) metric. When signal was driven by 100% of the species in the matrix, type II error rates were equal to or less than 0.05 for both NN and SDNN regardless of the randomization scheme (AWTS, AITS). Using the Abundance Weighted Trait shuffling method with the NN metric resulted in acceptable type II error rates (< 30%) for matrices with larger numbers of plots even when the proportion of species contributing signal was as low as 20%. The matrices for which this applied (i.e. at 20% proportion) were restricted to those with 75 or more plots and only for NN.

AWTS in combination with SDNN was further restricted to cases of low species numbers (between 10 and 20 species). Abundance Independent Trait Shuffling (AITS) only produced acceptable type II error rates at 60% contribution in combination with SDNN and was limited to 10 species and from 14 to 50 plots. Type II error rates at 80% contribution were limited to 15 and fewer species for NN and 10 and 15 species for SDNN. The minimum number of plots ranged from seven (AITS & NN; species = 10) to 30 (AITS & NN; species = 15) (Figure 3-2).

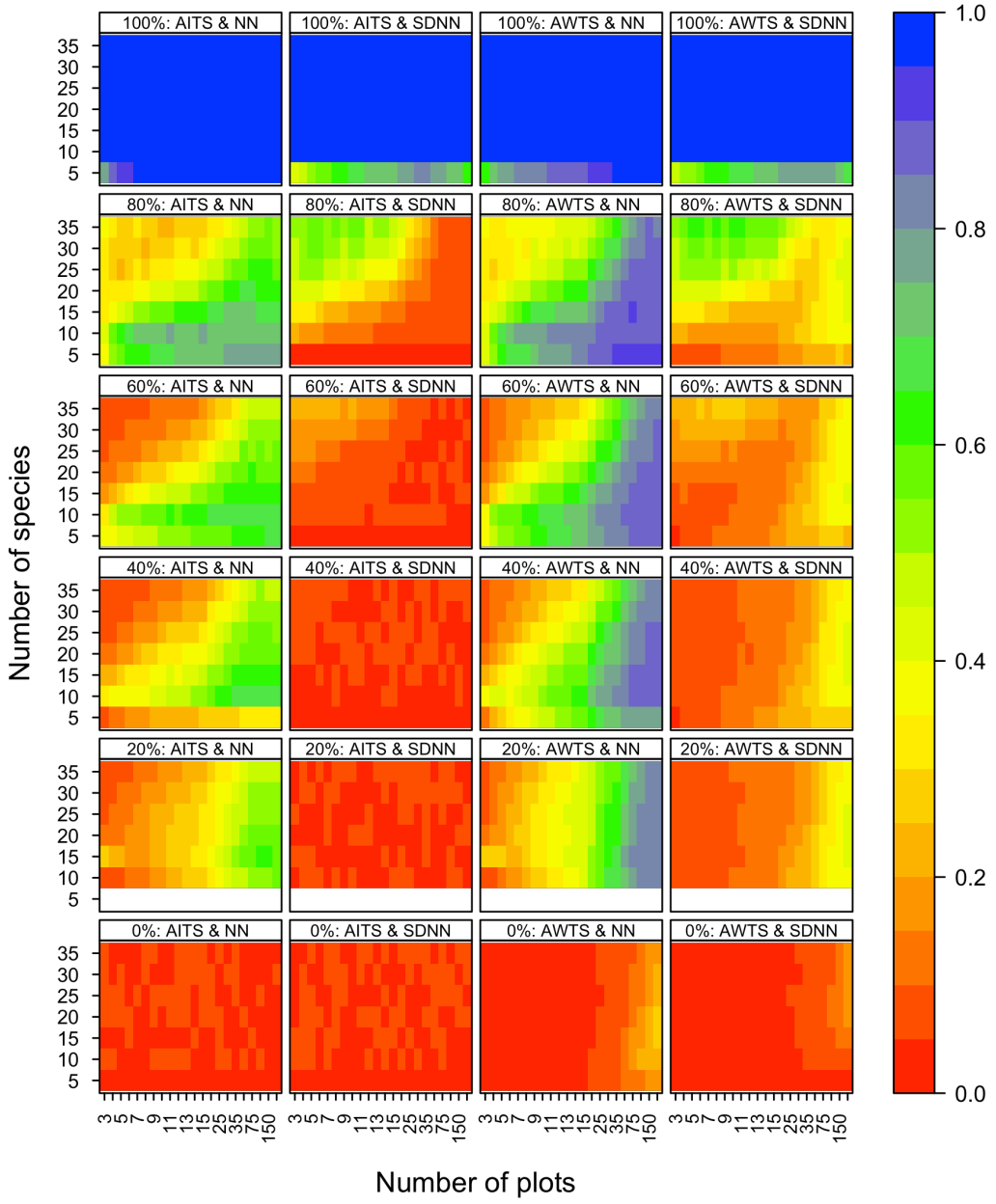

Figure 3-2. The power of the limiting similarity null models (1 – β) with respect to matrix dimension (number of species × number of plots) and the proportion of species contributing signal to the pattern of limiting similarity. Statistical power was estimated by maximizing the mean nearest neighbour distances (NN) and minimizing the standard deviation of the nearest neighbour distances (SDNN) metrics for set proportions of species prior to running the null model. The ability of the null models to detect the amplified patterns of limiting similarity were determined using a *p*-value of 0.05 (one-tailed). Matrices were swapped 5000 times using either Abundance Independent Trait Shuffling (AITS) or Abundance Weighted Trait Shuffling (AWTS) (Dante et al. 2013). Each cell within

the plots represents the proportion of 10,000 null models that detected significant limiting similarity. The percentage header for each plot represents the proportion of species contributing signal to the pattern of limiting similarity. Blue shading indicates high statistical power (low type II error rates) and red shading indicates low statistical power (high type II error rates). White cells indicate no data and are associated with proportions that result in single species for which it is impossible to maximize trait metrics. The plot of 0% species proportion is equivalent to a type I error rate estimation and acted as a negative control for our method of analysis.

Both AITS and AWTS in combination with NN and SDNN resulted in some *p*-values greater than 0.05 when no species were contributing signal (proportion = 0%) however, only AWTS resulted in error rates greater than 0.10. Error rates increased when the number of plots were greater than 100 for AWTS and SDNN and greater than 35 for AWTS and NN. Both randomization schemes and metrics showed improved type II error rates as the number of plots increased and the number of species decreased.

3.5 Discussion

The results of this study indicate that two common approaches to test for patterns of null model negative co-occurrence and limiting similarity suffer from increased type II errors when only subsets of species within the community contribute signal. This is a strong indicator that the lack of support for limiting similarity in the literature (Götzenberger et al. 2012) may arise from the failure of our tests to detect the pattern. It also indicates that, even though negative co-occurrence is found more often than limiting similarity, it too may be under represented by null model tests. Although more support for negative co-occurrence than for limiting similarity has been found (Gotelli and McCabe 2002, Götzenberger et al. 2012) this may simply be an artefact of the differential performance of the null models across community dimensions. That is, when a community is made up of few species (e.g., Stubbs and Wilson 2004, Wilson and Stubbs 2012), the likelihood of finding patterns of limiting similarity are high and finding those of negative co-occurrence are low. As the number of species increases to 15 or more, the likelihood of pattern detection rapidly decreases for limiting similarity and increases for negative co-occurrence. This favours finding significant negative co-occurrence over limiting similarity which is consistent with Götzenberger *et al.* (2012).

3.6 Conclusion

This study was carried out to assess type II error rates of negative co-occurrence and limiting similarity null models when only a proportion of species are contributing signal to the patterns being sought. We found that type II error rates for both classes of null model increased rapidly as the proportion of species contributing signal decreased. That is, the null models were not able to reliably detect patterns of limiting similarity or negative co-

occurrence when the proportion of species contributing signal fell below 80%. While null models may be able to correctly identify significant patterns of limiting similarity and negative co-occurrence at the whole community level (i.e. when the majority of species are involved in the pattern), they are unable to provide insight into the intricacies of community assembly. If only subsets of species in the community are contributing to patterns of community assembly, then these null models are unlikely to provide a clear picture of the community assembly processes at work.

The granular nature of species interactions, and as a result community assembly processes, requires a method of assessment that is also granular in nature. These null models, limiting similarity and negative co-occurrence, are unable to provide the level of granularity needed to truly understand community assembly and at this time there appears to be no easy solution to the problem. Ecologists still need tools that will provide granular insight into the granular nature of community assembly.

3.7 References

Cornwell, W. K. and Ackerly, D. D. 2009. Community assembly and shifts in plant trait distributions across an environmental gradient in coastal California. - Ecological Monographs 79: 109–126.

Cornwell, W. K. et al. 2006. A trait-based test for habitat filtering: Convex hull volume. - Ecology 87: 1465–1471.

Dante, S. K. et al. 2013. Evidence of deterministic assembly according to flowering time in an old-field plant community. - Funct Ecol 27: 555–564.

de Bello, F. et al. 2009. Partitioning of functional diversity reveals the scale and extent of trait convergence and divergence. - Journal of Vegetation Science 20: 475–486.

Diamond, J. and Gilpin, M. 1982. Examination of the “null” model of Connor and Simberloff for species co-occurrences on Islands. - Oecologia 52: 64–74–74.

Fayle, T. M. and Manica, A. 2010. Reducing over-reporting of deterministic co-occurrence patterns in biotic communities. - Ecological Modelling 221: 2237–2242.

Fisher, R. A. and Yates, F. 1953. Statistical tables for agricultural, biological and medical research. - Edinborough: Oliver & Boyd.

Gause, G. F. 1932. Experimental studies on the struggle for existence I. Mixed population of two species of yeast. - J Exp Biol 9: 389–402.

Gotelli, N. J. 2000. Null model analysis of species co-occurrence patterns. - Ecology 81: 2606–2621.

Gotelli, N. J. and Abele, L. G. 1982. Statistical distributions of West Indian land bird families. - Journal of Biogeography 9: 421–435.

Gotelli, N. J. and Ellison, A. M. 2002. Assembly rules for New England ant assemblages. - Oikos 99: 591–599.

Gotelli, N. J. and McCabe, D. J. 2002. Species co-occurrence: A meta-analysis of J. M. Diamond's assembly rules model. - Ecology 83: 2091–2096.

Gotelli, N. J. and Rohde, K. 2002. Co-occurrence of ectoparasites of marine fishes: A null model analysis. - Ecology letters 5: 86–94.

Götzenberger, L. et al. 2012. Ecological assembly rules in plant communities--approaches, patterns and prospects. - Biol Rev Camb Philos Soc 87: 111–127.

Grime, J. P. 2006. Trait convergence and trait divergence in herbaceous plant communities: Mechanisms and consequences. - Journal of Vegetation Science 17: 255–260.

Hardy, O. J. 2008. Testing the spatial phylogenetic structure of local communities: Statistical performances of different null models and test statistics on a locally neutral community. - Journal of Ecology 96: 914–926.

Kraft, N. J. B. and Ackerly, D. D. 2010. Functional trait and phylogenetic tests of community assembly across spatial scales in an Amazonian forest. - Ecological Monographs 80: 401–422.

Kraft, N. J. B. et al. 2008. Functional traits and niche-based tree community assembly in an Amazonian forest. - Science (New York, N.Y.) 322: 580–582.

Maestre, F. T. and Reynolds, J. F. 2007. Amount or pattern? Grassland responses to the heterogeneity and availability of two key resources. - Ecology 88: 501–511.

Maltez-Mouro, S. et al. 2010. Co-occurrence patterns and abiotic stress in sand-dune communities: Their relationship varies with spatial scale and the stress estimator. - Acta Oecologica 36: 80–84.

Mayfield, M. M. and Levine, J. M. 2010. Opposing effects of competitive exclusion on the phylogenetic structure of communities. - Ecology letters 13: 1085–1093.

Mouillot, D. et al. 2007. Limiting similarity, niche filtering and functional diversity in coastal lagoon fish communities. - Estuarine, Coastal and Shelf Science 71: 443–456.

Ribichich, A. M. 2005. From null community to non-randomly structured actual plant assemblages: Parsimony analysis of species co-occurrences. - Ecography 28: 88–98.

Rooney, T. P. 2008. Comparison of co-occurrence structure of temperate forest herb-layer communities in 1949 and 2000. - Acta Oecologica 34: 354–360.

Schamp, B. S. and Aarssen, L. W. 2009. The assembly of forest communities according to maximum species height along resource and disturbance gradients. - Oikos 118: 564–572.

Schamp, B. S. et al. 2008. Dispersion of traits related to competitive ability in an old-field plant community. - Journal of Ecology 96: 204–212.

Stone, L. and Roberts, A. 1990. The checkerboard score and species distributions. - Oecologia 85: 74–79.

Stubbs, W. J. and Wilson, J. B. 2004. Evidence for limiting similarity in a sand dune community. - Journal of Ecology 92: 557–567.

Veech, J. A. 2013. A probabilistic model for analysing species co-occurrence. - Global Ecology and Biogeography 22: 252–260.

Weiher, E. et al. 1998. Community assembly rules, morphological dispersion, and the coexistence of plant species. - Oikos 81: 309–322.

Wilson, J. B. and Stubbs, W. J. 2012. Evidence for assembly rules: Limiting similarity within a saltmarsh. - Journal of Ecology 100: 210–221.

Zhang, J. et al. 2009. Fine-scale species co-occurrence patterns in an old-growth temperate forest. - Forest Ecology and Management 257: 2115–2120.

Chapter 4.0 Preamble

The previous two chapters in this thesis have established that the negative co-occurrence and limiting similarity null models have acceptable type I and type II error rates and that they have poor statistical power when the proportion of species contributing signal to patterns each fall below 80%. This chapter builds on these results first by developing a method for the detection of patterns of limiting similarity and negative co-occurrence when signal is weak (i.e. when the proportion of species contributing signal is less than 80%). It then tests the ability of three measures of association and two measures of pairwise co-occurrence to correctly identify species that are contributing signal to the patterns of limiting similarity and negative co-occurrence.

In this chapter, synthetic matrices were created for a broad range of community dimensions. Known proportions of randomly selected species had patterns of either limiting similarity or negative co-occurrence added. Null model tests were carried out on 1000 subsets of species for each subset size tested. The subset sizes tested were {5, 10, 15, 20, 25, 50} and all subset sizes were tested when the number of species in the community allowed (i.e., subset size < the total number of species). The *p*-value of each null model test was recorded along with the subset of species tested. The occurrence of significant pattern within a subset of species was determined by counting the number of null model tests with a *p*-value > 0.95 and the null hypothesis rejected at $\alpha = 0.05$ (H_0: There is no pattern of limiting similarity/negative co-occurrence in the community).

The data and matrices generated from the subsetting tests were used for the association tests and pairwise co-occurrence tests. The three species association tests evaluated were frequent pattern mining, indicator species analysis and indicator species analysis using species combinations. The two measures of species pairwise co-occurrence tested were the probabilistic model (Veech 2013) and 95% confidence limit criterion. The ability of each method to correctly identify the species contributing signal was determined by calculating the intersection and complement of the set of species with signal added and the set of species returned from the method.

The results of this chapter demonstrate that the method of subsetting species is able to detect patterns of limiting similarity or negative co-occurrence when the proportion of species in the community that were contributing signal was less than 80%. It also demonstrated that frequent pattern mining in conjunction with the subsetting of species, has potential as a tool to identify species contributing signal to patterns of limiting similarity and negative co-occurrence.

4.0 Using species subsets with limiting similarity and negative co-occurrence null models enhances detection of community assembly

4.1 Abstract

Limiting similarity and negative co-occurrence are community patterns that ecologists regularly employ a suite of null models to explore. While each class of null model is able to detect their respective patterns when 80% of species in a community are involved, neither is able to provide insight into the details of community assembly when a lower proportion species in the community are involved. New tools are needed to detect and describe the presence of these patterns in communities. We develop a novel two-step approach aimed at illuminating the intricacies of community assembly at the sub-community level. The first step is an extension of well-established null model approaches to with-community assembly patterns by testing random subsets of species within the community for significant pattern. The second step explores the utility of four methods: indicator species analysis, the probabilistic model of co-occurrence, the 95% confidence limit criterion, and frequent pattern mining, to identify the groups of species adding signal to the matrices. We used benchmark analysis using synthetic matrices to estimate type I and type II error rates for the subsetting method and an exploratory approach to assess the ability of each of the four methods to correctly identify the species with signal added. Benchmarking indicates that the subsetting of species maintains acceptable type I and type II error rates and that no artefactual signal is introduced by the subsetting method. Tests of species associations within significant subsets were inconclusive, likely due to a significant number of background associations that appear to occur natively in the randomly generated matrices. This method of assessing limiting similarity and negative co-occurrence provides ecologists with a powerful tool for assessing whether these patterns are present in a set or sets of species within the community and not just predominant at the whole community level.

4.2 Introduction

Community ecologists have long been interested in the process of community assembly with a goal of classifying and quantifying the individual mechanisms involved (Levin 1970; MacArthur 1970; Diamond 1975; Weiher & Keddy 1999; Hubbell 2001; 2005; Rosindell *et al.* 2011; Helsen *et al.* 2012). One mechanism, competition, has garnered a great deal of attention over the years with patterns of species negative co-occurrence or limiting similarity hypothesized as possible outcomes. The concept of species negatively co-occurring originated with Gause's (1932) competitive exclusion principle, which postulates that species that compete for limited resources cannot co-exist. Limiting similarity, on the other hand, has its foundation in the work of MacArther & Levins (1967) and theorizes that species are able to avoid competition and co-exist as the result of differences in trait values. Both theories have been used to assess communities for patterns believed to result from competition; however, there still exists considerable uncertainty about the role that each plays in structuring communities (Götzenberger *et al.* 2012).

Two classes of null model, limiting similarity and negative co-occurrence, are commonly used in studies aimed at identifying patterns of community assembly (Weiher *et al.* 1998; Gotelli & Rohde 2002; Gotelli & Ellison 2002; Ribichich 2005; Maestre & Reynolds 2007; Rooney 2008; Zhang *et al.* 2009; Maltez-Mouro *et al.* 2010); limiting similarity: (e.g., Weiher *et al.* 1998; Stubbs & Wilson 2004; e.g., Cornwell *et al.* 2006; Mouillot *et al.* 2007; Hardy 2008; Schamp *et al.* 2008; Kraft *et al.* 2008; de Bello *et al.* 2009; Schamp & Aarssen 2009; Kraft & Ackerly 2010; Wilson & Stubbs 2012); however, they do so in different ways. Co-occurrence analyses evaluate the existence of segregating, or negatively co-occurring species pairs against null expectations whereas limiting similarity evaluates whether trait distributions among coexisting species are more variable than expected under a null model. In the case of limiting similarity, if co-occurring species differ in trait values more often than expected under the null model, this is generally interpreted as evidence that competition is segregating species that significantly overlap in niche. For the co-occurrence null models interpretation is more straightforward with greater negative co-occurrence than expected under the null model suggesting that competitive exclusion is segregating

species that compete asymmetrically. When species positively co-occur, this suggests facilitation among species or abiotic filtering.

One of the key assumptions with the use of these null models, as with any statistical analysis, is that they perform with acceptable and predictable type I and type II error rates. While the initial work investigating the statistical performance of limiting similarity and negative co-occurrence null models indicates that both classes of null model perform with acceptable error rates (Gotelli 2000; Hardy 2008 Chapter One), it is evident that both null models are unlikely to identify relevant patterns of co-occurrence or limiting similarity when only a proportion (subset) of species in the community are contributing signal to the pattern being sought (Chapter Three). This poses a significant problem when researchers are interested in understanding whether competition is contributing to patterns of negative co-occurrence and limiting similarity in general, rather than the community-scale analog, which can only test whether these patterns are dominant in the focal community. This difference may also explain why limiting similarity has found limited support in studies aimed at detecting it (Götzenberger *et al.* 2012) (but see Stubbs & Wilson 2004; Wilson & Stubbs 2012) when other evidence suggests that these patterns may still be relevant either for subsets of species, or for broad groups of species (McKane *et al.* 2002). The challenge may not be as substantial for patterns of negative co-occurrence; negative co-occurrence has been found more often than limiting similarity (Götzenberger *et al.* 2012), perhaps indicating that this tends to be a dominant pattern in many natural systems. Nevertheless, based on the results described in Chapter Three, negative co-occurrence has the potential to benefit from a method that can uncover patterns of co-occurrence when they are not dominant in a community, and identify groups of species for which the pattern is strong.

Given that limiting similarity and negative co-occurrence null models both have acceptable type I error rates ($\alpha \leq 0.05$) for a broad range of matrix dimensions (Chapter Two) and that they also have acceptable type II error rates ($\beta <= 0.20$; (Cohen 1992)) when greater than 80% of the species in the community contribute signal to the pattern sought (Chapter Three), we focus here on testing the efficacy of potential tools for identifying

these patterns within communities when the signal is not as powerful, and for identifying the species that contribute to these patterns within communities.

4.3 Methods

To test the effectiveness of subsetting, the random selection of groups of species from within the community, at detecting limiting similarity and negative co-occurrence with weak signal we used a two-step approach. The first step was to assess type I and type II error rates of the subsetting method on matrices that only had a few species contributing signal (weak signal) to the pattern. The second step, a validation step, served two purposes: 1) to determine if subsetting was detecting patterns in the correct groups of species (i.e. in the species that we set up to contribute signal) and 2) to test the ability of subsetting to provide detailed information on the community assembly process.

4.3.1 Presence-absence matrices

Matrices were generated using the same methods as in Chapter One and Chapter Two and is repeated here for completeness. Presence-absence matrices of *m* rows by *n* columns were generated by randomly assigning individuals to matrix cells. The incidence, N_i, of each species (the number of plots that a species occurred in) was determined by sampling from a log-normal distribution ($N_i = e^{x_i/2a}$) where $x_i \sim N(0, 1)$ and a is a shape generating parameter (Ulrich & Gotelli 2010). The shape-generating parameter a was set to 0.2 as that value is appropriate for large, well sampled communities (Preston 1962a; b). The log-normal distribution was chosen over the power series and log series distributions due to its combined fit to both well sampled and incompletely sampled communities (Ulrich *et al.* 2010). The resulting incidence values were converted to integers (from decimal values) by truncation and values equal to zero or greater than the number of plots (n) discarded. This was done as presence-absence data are binary in nature and row totals could only be integer values. The removal of zero values was required to prevent the generation of degenerate matrices with species that do not occur in any plots (Gotelli 2000).

Species were assigned to plots with the probability of assignment proportional to the number of incidences of that species divided by the number of plots. Species presences were assigned this way until all incidences were accounted for and all plots had at least one

species present. If the resulting matrix contained a plot with no species present, the matrix was discarded and the whole process was re-started. This resulted in matrices that had at least one occurrence for each species and at least one species present in each plot.

Other approaches to matrix generation could have been used to generate community data, such as a neutral model (Bell 2000); however, our approach does not presuppose an underlying ecological mechanism and is consistent with previous approaches (Gotelli 2000; Ulrich & Gotelli 2010). The resulting communities contained log-normal species abundance patterns such as those commonly observed in natural systems.

4.3.2 Null models and metrics

For co-occurrence tests, we used a single null model, the fixed-fixed independent swap algorithm (Connor & Simberloff 1979; Gotelli 2000), in combination with the C-Score (Stone & Roberts 1990) . This randomization algorithm and co-occurrence metric were chosen due to their prevalence in the literature and previous validation as a reliable measure of co-occurrence (Gotelli 2000).

For limiting similarity tests the abundance independent trait shuffling randomization algorithm (AITS: see Dante *et al.* 2013) was used in combination with two metrics: Nearest Neighbour (NN) distances and, Standard Deviation of Nearest Neighbour distances (SDNN). Abundance weighted trait shuffling, another commonly used randomization algorithm, was not used for matrix randomization as it was not as statistically robust as the AITS algorithm (Chapter Two; Chapter Three). The AITS method of shuffling trait values between species without constraint has been commonly employed in previous studies (e.g., Kraft *et al.* 2008; Schamp & Aarssen 2009; Cornwell & Ackerly 2009; Dante *et al.* 2013) and has been shown to have good statistical performance (Chapter Two, Chapter Three).

4.3.3 Generating pattern

To test the ability of the null models to detect patterns of co-occurrence or limiting similarity, we needed to create matrices that contained elevated amounts of signal in a known proportion of species. We created matrices with no significant pattern using the

methods above and then elevated the signal for patterns of negative co-occurrence or limiting similarity in a controlled manner. While the method used to create the matrices did not guarantee that each matrix was void of significant pattern the probability of producing a matrix with significant pattern was less than five percent (Chapter Two). The approach that we used to introduce signal was based on the methods used by Lavender (Chapter Three) and is repeated in the following sections.

4.3.3.1 Limiting Similarity

Because one of the expectations under limiting similarity is that coexisting species are more likely to have significantly different trait values, the first step was to group species by pairwise co-occurrence. To carry out this separation we used a pairwise test of species co-occurrence (Veech 2013) with a *p*-value of 0.05. Patterns of limiting similarity were added to proportions of species within the matrices by randomly selecting species from the group of neutrally/positively co-occurring species as determined using Veech's method above. Synthetic matrices consisting of all combinations of *m* species, *m* = {10, 15, 20, 40, 60, 80, 100}, by *n* plots, *n* = {25, 50, 100, 150, 200} were used. The number of species selected to contribute signal to matrices were n_{signal} = {0, 5, 10, 15, 20}. The values for n_{signal} were selected to test the ability of subsetting to detect patterns when signal was weak (<= 20%). We used the case of n_{signal} = 0 (i.e. no pattern added) as a negative control for increased type I errors which may have arisen as a result of the subsetting method itself. Trait values were assigned to n_{signal} species by drawing trait values at random from a uniform distribution and assigning them to species sequentially. After all species in the n_{signal} group had been assigned trait values, the metrics NN or SDNN were calculated for the n_{signal} group and stored. This was repeated 100,000 times, selecting n_{signal} new species and trait values with each repetition. With each re-assignment, if the new values for NN or SDNN were greater than the stored values, the new species and trait combinations were stored, resulting in maximal values for the test metrics. Once these maximal values were determined for the group of species contributing signal, the limiting similarity null model was run on the full matrix which contained known proportions of introduced, non-random trait structure.

4.3.3.2 Negative co-occurrence

The approach used to add pattern to the matrices for negative co-occurrence was similar to that used for limiting similarity. For the co-occurrence null model tests, we generated matrices for all combinations of *m* species, *m* = {10, 15, 20, 40, 60, 80, 100}, by *n* plots, *n* = {20, 40, 60, 80, 100}. Patterns of negative co-occurrence (non-random patterns) were added to presence-absence matrices without significant negative co-occurrence in a controlled manner by selecting a subset of n_{signal} species at random from each matrix where n_{signal} = {0, 5, 10, 15, 20}. To produce subsets of species with maximal C-Scores (i.e., high negative co-occurrence), the C-Score was first determined for the subset after which each species was re-assigned among the plots by using a Fisher-Yates shuffle (Fisher & Yates 1953). This method of shuffling the subset did result in matrices with different species richness for some plots as well as some plots containing no species. We did not, however, adjust the matrix around the placement of the non-random species (i.e. those that we maximized the C-Score for) to maintain species richness/plot, as this could have added structure elsewhere in the matrix. Once all species in the subset had been shuffled among plots the C-Score was re-calculated. If the new C-Score was greater than the previous value then the new matrix was stored. This process was repeated 100,000 times on that species subset in an attempt to maximize the C-Score for the subset under consideration. This approach ensured that each matrix had a subset of species that were approaching maximal negative co-occurrence as a group.

4.3.4 Subsetting Tests

4.3.4.1 Limiting similarity

Subset size (*s*) was the number of species selected at random from the matrix for null model analysis, where *s* = {5, 10, 15, 20, 25, 50}. All subset sizes were used with each matrix allowing us to determine if subset size was important for the detection of significant patterns. One exception to this occurred when the subset size would have been larger than the total number of species in the matrix (*s* > *m*), which made it impossible to include that subset size. As a positive control we used the case where signal was added to all species in the community and the subset size (*s*) was equal to the number of species ($m = s = n_{signal}$).

For each combination of matrix dimension and number of species contributing signal, we generated 100 synthetic matrices. Then for each matrix and subset size (s) we randomly selected 1000 subsets of size s from the species within the matrix. The limiting similarity null model was run on every subset generated. For each null model test we kept track of significance ($\alpha < 0.05$) as well which species were included in the subset. This was repeated for both limiting similarity metrics (NN and SDNN). All tests of significance were one-tailed due to the fact that our method of adding structure was unidirectional (i.e. it increased limiting similarity) and significance was determined using exclusive p-values (Chapter Two).

4.3.4.2 Negative co-occurrence

The methods used to subset and test the negative co-occurrence null model were identical to those used for the limiting similarity except for the following: signal added to the synthetic matrices was in the form of negative co-occurrence, the negative co-occurrence null model was used instead of the limiting similarity null model (see *Null models* above) and, the maximum number of plots (n) was reduced from 200 to 100 due to the significantly longer run times required for the negative co-occurrence null model relative to the limiting similarity null model. The parameters used were; species: $m = \{10, 15, 20, 40, 60, 80, 100\}$, plots: $n = \{20, 40, 60, 80, 100\}$, and subset sizes: $s = \{5, 10, 15, 20, 25, 50\}$.

For each combination of matrix dimension ($m \times n$) and number of species contributing signal (n_{signal}) we generated 100 synthetic matrices. Then for each matrix and subset size (s) we randomly selected 1000 subsets of size s from the species within the matrix and ran the negative co-occurrence null model on each keeping track of the null models significance the list of species included in the subset. As with our limiting similarity tests, all tests of significance were unidirectional (i.e. it increased negative co-occurrence) and significance was determined using exclusive p-values (Chapter Two).

4.3.5 Verification of significant subsets

Because no previous work has examined methods for identifying species subsets in this manner we also investigated the ability of five methods to identify species associated with significant pattern and by doing so verify that subsetting detects the correct group of

species in the matrices (i.e. pattern coming from the species that signal was maximized for). The methods tested included: Indicator Species analysis (Dufrêne & Legendre 1997), species combinations with Indicator Species analysis (De Cáceres *et al.* 2012), frequent pattern mining using the FP-Growth algorithm (Han *et al.* 2000), a probabilistic model of assessing species co-occurrence (Veech 2013) and the 95 percent confidence limit criterion (Sfenthourakis *et al.* 2004). The probabilistic model and 95 percent confidence limit criterion were only used with co-occurrence data, as they are specific to that type of data; the other three have the advantage of being more broadly applicable. These analyses were run on the data that were previously generated for the subsetting tests above. Beyond verification of significant subsets, these methods could provide a critical tool to the researcher investigating limiting similarity and co-occurrence in subsets of a community by identifying the species in the significant subset.

4.3.5.1 Indicator species

For the indicator species analysis, each of the 1000 randomly selected subsets was treated as a list of species present at a site and the sites were categorised into one of two groups: subsets that yielded significant negative co-occurrence results, and those that did not. The indicator value used was the group-equalized index for presence-absence data and the statistic used for the permutation test (null model test) was A_{pa}^{g} (De Cáceres & Legendre 2009). All indicator species that resulted in a z-score > 1.93 were added to the list of species considered to be adding signal to the matrix (or observed list of species). We selected 200 permutations for the permutation tests as this was the value used by De Cáceres & Legendre (2009) and, in trial runs, greater numbers of permutations had no influence on the outcome of the tests but greatly increased processing time.

4.3.5.2 Indicator species – species combinations

The use of species combinations in indicator species analysis (De Cáceres *et al.* 2012) was considered as a method of analysis; however, it requires *a priori* knowledge of species groups to be used efficiently. While this method may have potential, the processing time for a single subset in tests runs ranged from several hours to days. Consequently, testing all

possible combinations of species groupings was not practical and this method was not pursued further.

4.3.5.3 Frequent pattern mining

Frequent pattern mining is a method of analysing shopping basket data for association rules and was first developed for this purpose by Agrawal et al. (1993). The intent of the analysis was to analyse consumer buying behaviours by looking for associations between items that customers placed in their "shopping basket". The approach was designed to determine how likely it was that customers purchasing bread, for example, also purchase butter and/or milk at the same time. The general approach used for this type of analyses is to "mine" a database of cash register receipts (transactions) looking for groups of items (itemsets) that occur together at some user specified minimum frequency (minimum support) resulting in a list of "frequent itemsets" (see Han *et al.* 2007 for an overview of frequent pattern mining and its current state).

For the purposes of our study, instead of looking for purchasing behaviours, we used frequent pattern mining to look for associations between significant null model results and the species contributing signal. We also limited the frequent pattern mining analyses to discovery of all frequent two-species itemsets (i.e. species pairs). The data mining procedure used to search the significant subsets of frequently occurring groups of species was the FP-Growth algorithm (Han *et al.* 2000). Frequent pattern mining requires that the user provide a minimum support value for the analysis. The minimum support value is the minimum number of transactions (or in our case significant subsets) that an item (species) must occur in to be included in a frequent itemset. For example, if the subsetting method resulted in 100 significant subsets for limiting similarity and we set the minimum support value to 50%, we would filter out any single species itemsets that occurred in fewer than half of the significant subsets. In the next iteration of the process, when mining two-species itemsets, the minimum support is kept the same but is relative. That is, if we obtained a single species itemset {sp01} that occurred 50 times out of 100, we would then keep all two-species itemsets containing sp01 where the second species occurred at least 25 times out of 50 (50 × 0.5). Because there is currently no objective method for selecting a minimum support level in frequent pattern mining, we opted to test

the algorithm using a minimum support level of 30%. This level was selected based on its ability to produce frequent pattern data across the broad range of matrices and subset sizes tested. With minimum support levels of less than 20% we ran the risk of extremely long run times and insufficient computing resources (i.e., memory and processing power). With higher minimum support values (greater than 50%) the number of frequent patterns returned decreased rapidly with many matrices returning no frequent patterns. Species identified in these analyses were stored as a list of species believed to be contributing signal to the focal matrix and this list was then compared to the original list of species to which signal had been added.

4.3.5.4 Probabilistic model of species co-occurrence

The probabilistic model of Veech (2013) was also used to try and identify species contributing signal. This method is a pairwise comparison of species co-occurrence and uses the full presence-absence matrix that we added signal to as input. The output from this analysis is a list of all species pairs with an associated *p*-value. The *p*-value returned was then used to assess significance ($\alpha < 0.05$). The assumption with this test, and the next, is that the group of species that we had added signal to would be more likely to negatively co-occur with each other than with other species in the community. By combining all pairs of significantly negatively co-occurring species from the test we should be able to identify the list of species that we had originally added signal to.

4.3.5.5 95% confidence limit criteria

The 95% confidence limit criterion (Sfenthourakis *et al.* 2004) is another pairwise comparison method of assessing co-occurrence between species. This method takes the full presence-absence matrix as input and returns a list of all pairwise combinations of species, each with an associated *p*-value, which was used to assess the statistical significance of co-occurrence patterns. As with the probabilistic model, pairs of species with significant negative co-occurrence ($\alpha < 0.05$) were assumed to belong to the group of species that we had originally added signal to and were retained for comparison to the original list.

4.3.5.6 Evaluation of results

For all of the methods used for verification of the subsets: indicator species analysis, frequent pattern mining, 95% confidence limit criterion, and the probabilistic model of species co-occurrence, we assumed that the lists of species returned by each method should closely match the list of species that had signal added to them. To evaluate the effectiveness of each method to recover the original list of species with added signal, we calculated a single metric. This metric (matches) was the number of species in the intersection between the list of species with added signal (*A*) and the list of species returned by the method (*B*) over the number of species with added signal. The matches were calculated for each combination of matrix dimension, subset size, number of species with added signal, and null model test and were interpreted visually after plotting. To assess the amount of background signal in the matrices we also evaluated the number of species returned from matrices that had no signal added. These data were plotted alongside the match data for interpretation.

4.3.6 Software

All code for the null model analyses were written in Scala (Version 2.9.2) [Computer Language], available from http://www.scala-lang.org/downloads using IntelliJ IDEA Community Edition (Version 13.1.2) [Computer program], retrieved from http://www.jetbrains.com/idea/download/index.html and run on the Java VM (Version 1.7) [Computer software], available from http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html. Statistical analyses (non null model) were carried out using the R Project for Statistical Computing (Version 3.0.3) [Computer software], available at http://cran.r-project.org. All code used for this project is available in Appendix A and Appendix H as well as online at https://github.com/lavendermi

4.4 Results

4.4.1 Subsetting Tests

4.4.1.1 Limiting similarity

Running null model tests on subsets of species from community matrices enabled the detection of limiting similarity when it was present only when the mean nearest neighbour trait distance (NN) metric was used (Figure 4-1 & Figure 4-2). The standard deviation of the nearest neighbour distances (SDNN) metric rarely produced significant patterns when used with subsets (12 times, Figure 4-2). When no signal was present in the matrices (i.e. when no signal was added) subsetting did not detect any patterns of limiting similarity (Figure 4-1 & Figure 4-2). Subsetting resulted in greater than 95% detection rates for limiting similarity when the subset size was equal to the number of species in the community and all species were contributing signal to limiting similarity. This is consistent with Lavender et al. (Chapter Three) when the proportion of species contributing signal was equal to 100%.

The NN metric detected patterns of limiting similarity more often than the SDNN metric. For example, when NN was used as the metric of limiting similarity the null model was able to detect significant pattern with subset sizes as small as five when signal had been elevated in five and 10 species whereas, SDNN was not able to detect significant signal. This behaviour is consistent across all matrix dimensions, across the range of species with signal added, and across subset sizes, and is most obvious for the larger subset sizes (Sub=25 & Sub=50) (Figure 4-1 & Figure 4-2).

In general, subsetting performed better as the subset size approached the number of species in the community. The optimal range of subset size for the NN metric was between 25% and 100% of species (Figure 4-1). When signal was very weak (i.e. when only five species had elevated signal) NN was able to detect significant pattern regardless of the subset size used. The NN metric was able to detect pattern with subsets of five species however, it was only able to do so when the signal came from five and ten species and only when the community contained 15 or fewer species in total.

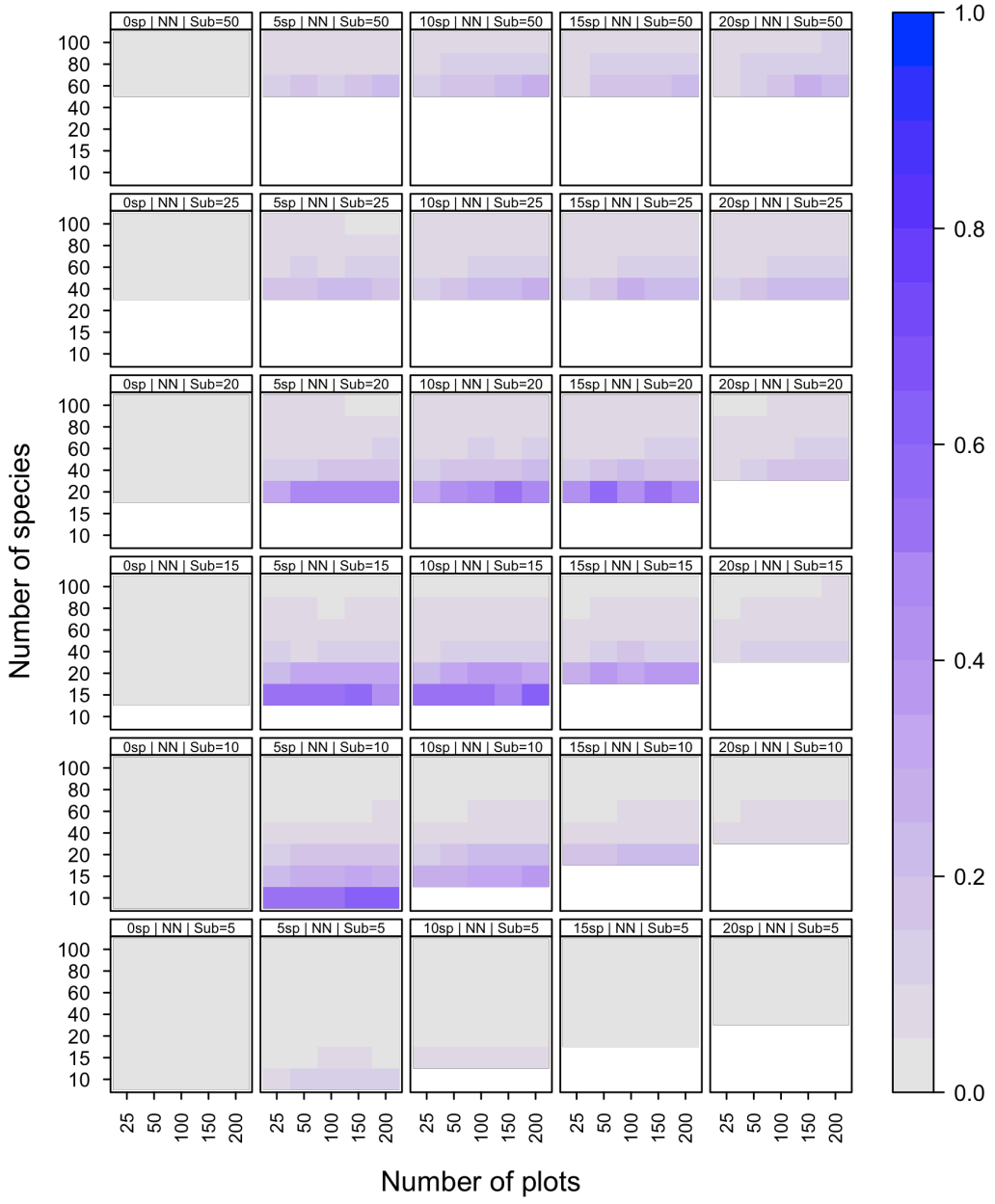

Figure 4-1. The proportion of subsets yielding significant patterns of limiting similarity with respect to matrix dimension, number of species contributing signal and subset size. Each cell within each plot represents the proportion of significant tests for 100,000 null models (100 matrices × 1000 subsets/matrix). The metric used for the limiting similarity null model was mean nearest neighbour trait distance (NN) and the randomization algorithm used was Abundance Independent Trait Shuffling (AITS) (see Dante *et al.* 2013). The plots are organized from left to right by the number of species contributing signal with no signal added in the left column (0sp) to a maximum of 20 species contributing signal along the right (20sp). The size of the subsets tested increases from bottom to top with

subsets of 5 species (Sub=5) being the smallest subsets tested and subsets of 50 species the largest (Sub=50). White space within the plots represent no data and are due to the fact that it was not possible to create subsets larger than the number of species in the matrix. Grey fill for a cell indicates fewer than 5% of the null models were significant and blue fill indicates greater than 95% of the null models were significant.

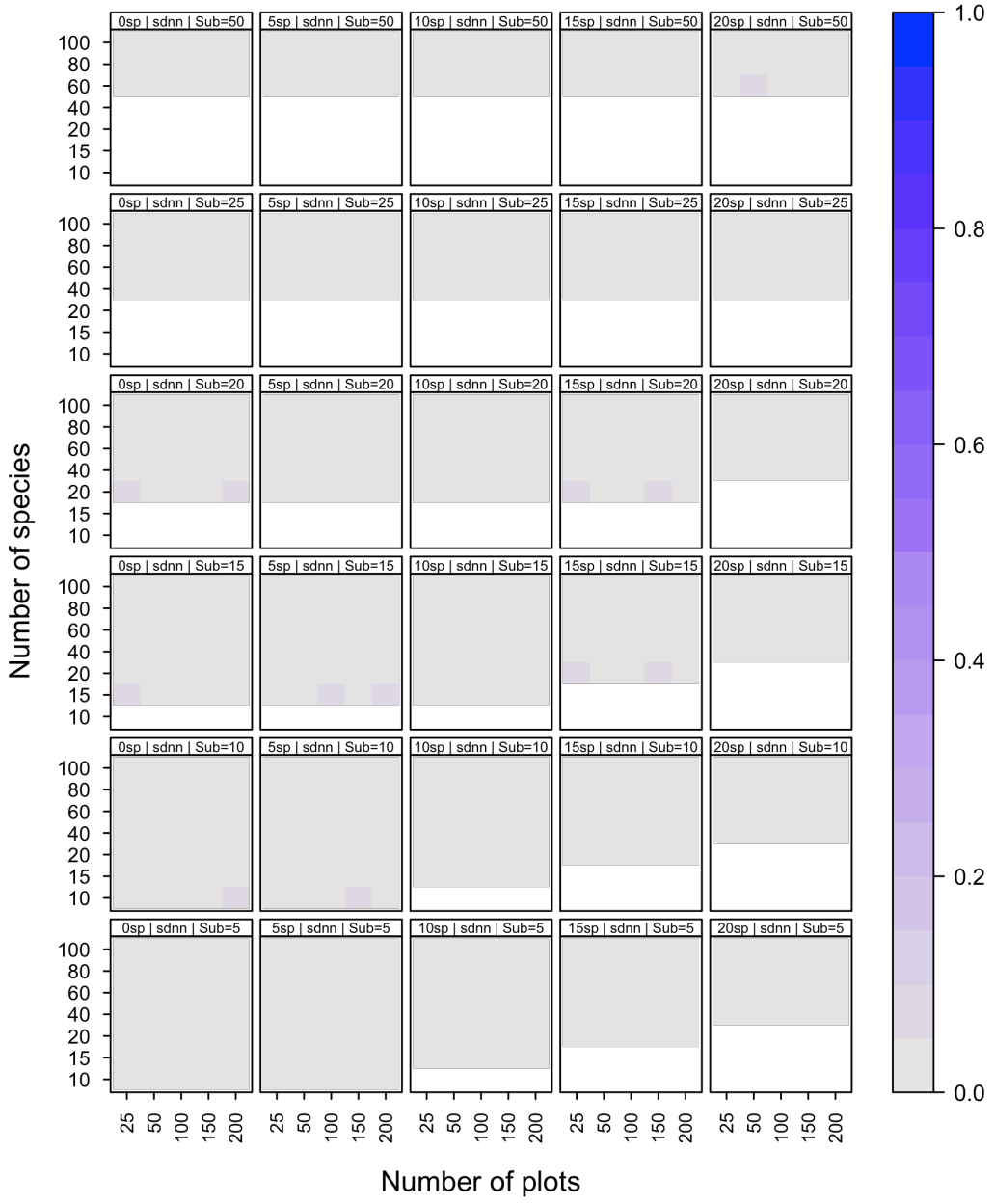

Figure 4-2. The proportion of subsets yielding significant patterns of limiting similarity with respect to matrix dimension, number of species contributing signal and subset size. Each cell within each plot represents the proportion of significant tests for 100,000 null models (100 matrices × 1000 subsets/matrix). The metric used for the limiting similarity null model was standard deviation of nearest neighbour trait distance (SDNN) and the randomization algorithm used was Abundance Independent Trait Shuffling (AITS) (see Dante *et al.* 2013). The plots are organized from left to right by the number of species contributing signal with no signal added in the left column (0sp) to a maximum of 20 species contributing signal along the right (20sp). The size of the subsets tested increases

from bottom to top with subsets of 5 species (Sub=5) being the smallest subsets tested and subsets of 50 species the largest (Sub=50). White space within the plots represent no data and are the due to the fact that it was not possible to create subsets larger than the number of species in the matrix. Grey fill for a cell indicates fewer than 5% of the null models were significant and blue fill indicates greater than 95% of the null models significant.

4.4.1.2 Negative co-occurrence

Subsetting of species prior to negative co-occurrence analyses had varied results with the best rates of detection obtained for matrices with 20 or fewer species in total (Figure 4-3) There was also a marked decrease in pattern detection rates with increasing plot number, consistent with Lavender et al. (Chapter Three). While subsetting with negative co-occurrence analysis did not, for the most part, detect signal in matrices with no added structure, it also failed to detect signal consistently for matrices with added structure, particularly as the number of species in the community increased.

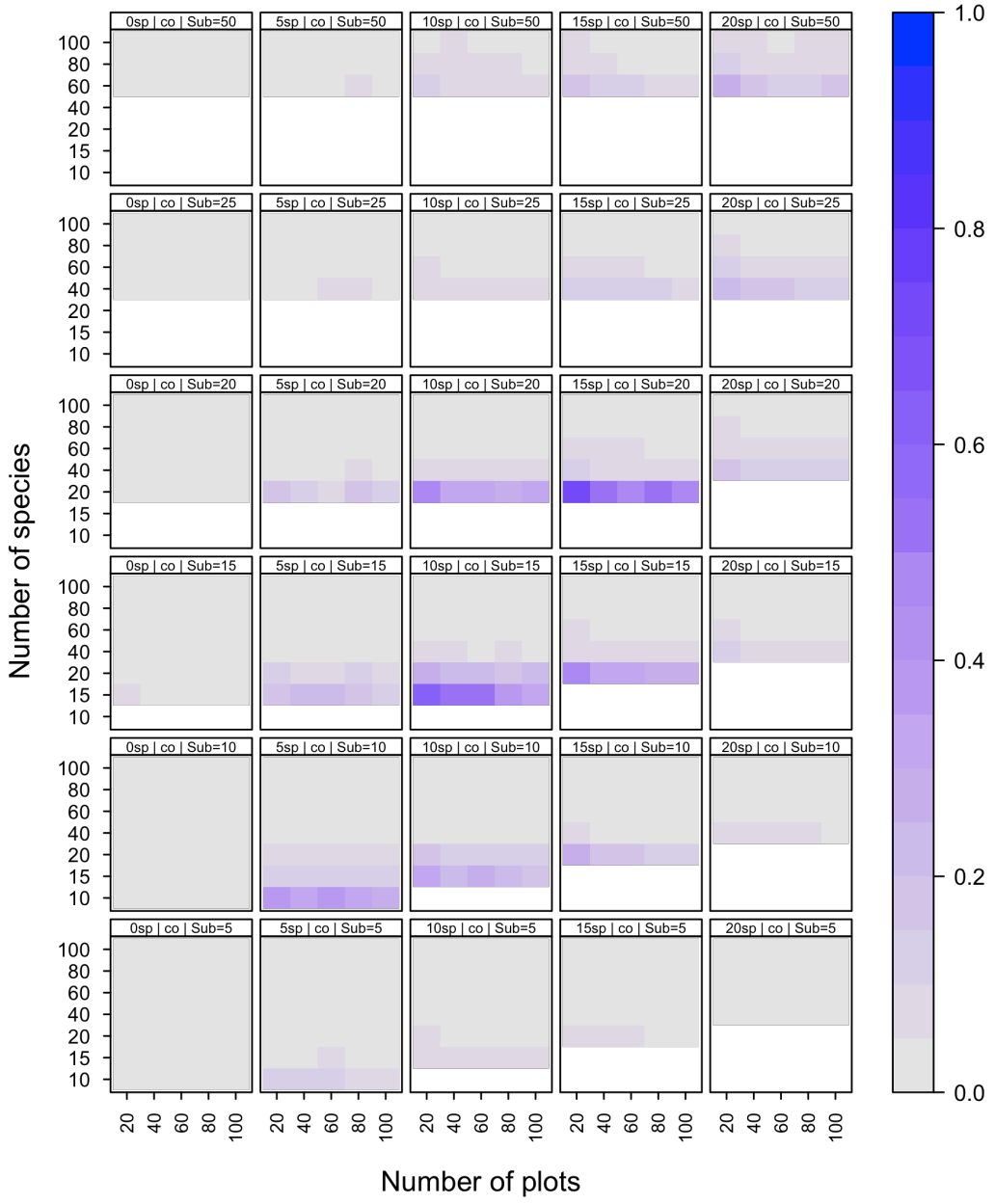

Figure 4-3. The proportion of subsets yielding significant patterns of negative co-occurrence with respect to matrix dimension, number of species contributing signal and subset size. Each cell within each plot represents the proportion of significant tests for 40,000 null models (40 matrices × 1000 subsets/matrix). The metric used for the negative co-occurrence null model was the C-Score (Stone & Roberts 1990) and the randomization algorithm used was the fixed-fixed, Independent Swap Algorithm (Gotelli 2000). The plots are organized from left to right by the number of species contributing signal with no signal added in the left column (0sp) to a maximum of 20 species contributing signal along the right (20sp). The size of the subsets tested increases from bottom to top with subsets of 5

species (Sub=5) being the smallest subsets tested and subsets of 50 species the largest (Sub=50). White space within the plots represent no data and are the due to the fact that it was not possible to create subsets larger than the number of species in the matrix. Grey fill for a cell indicates fewer than 5% of the null models were significant and blue fill indicates greater than 95% of the null models significant.

4.4.2 Verification of significant subsets

The results presented in the following sections are for the negative co-occurrence null model in combination with the four methods of species verification. We are presenting only the negative co-occurrence data as negative co-occurrence was the only null model that all four methods of species verification could be applied to as both the probabilistic model and 95% confidence limit method require binary presence-absence matrices as input and will not work with trait matrices. The standard deviation of the nearest neighbour trait distances (SDNN) metric was not used in the verification tests. This was due to the fact that it did not produce significant subsets for the subsetting tests. The results for both the co-occurrence and limiting similarity null models using the mean nearest neighbour trait distance metric (NN) are similar and as a result the comments made below for each method also apply to the limiting similarity null model unless otherwise noted. The full results for the data not presented here can be found in Appendix G for the plots associated with the limiting similarity null model.

4.4.2.1 Indicator species

Indicator species analysis indicated that a large number of species in the community are associated with significant patterns of negative co-occurrence even when no signal had been added to the matrices (Figure 4-4; see Figure G-1 and Figure G-2 in Appendix G for limiting similarity). Subset size had a strong influence on the number of species associated with negative co-occurrence with subset sizes equal to 20 to 30% of the total number of species in community resulting in the greatest number of species associated with negative co-occurrence. This pattern was evident in the limiting similarity data (Appendix G) as well. The ability of indicator species analysis to correctly identify the species that we had added signal to was strongly influenced by subset size. The lowest rate of identification occurred with a subset size of five and resulted in a range in match rates with between 40% and 80% of species being identified. Rates of up to 100% were achieved when subset sizes of 15, 20 and 25 were used and the subset size was between 20 and 30% of the total number of species in the matrix. The highest match rates for species with signal coincided with the highest numbers of species in the matrices with no signal added. That is, when the number of species associated with significant patterns in no signal added plots was equal to 100%.

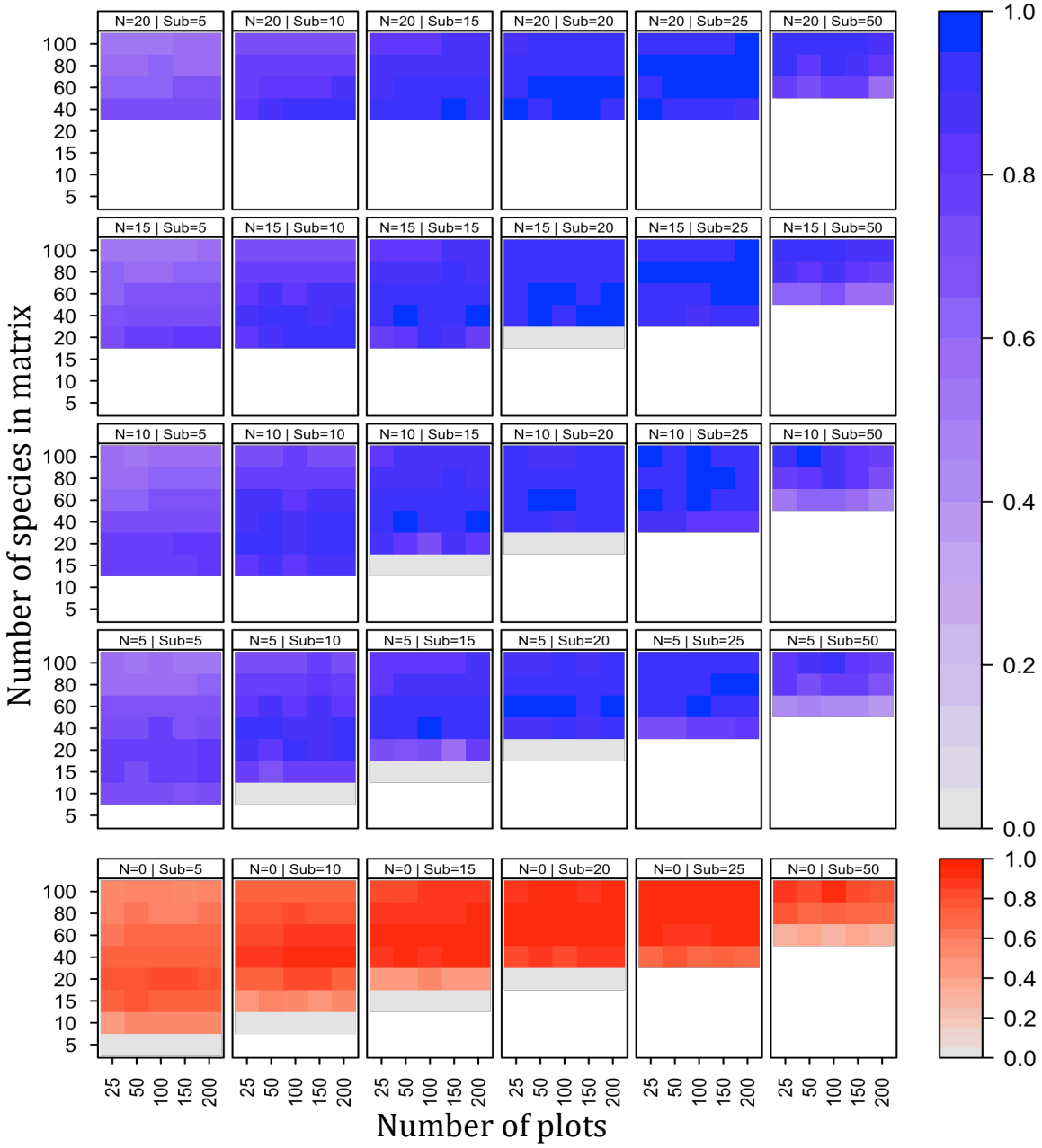

Figure 4-4. The mean proportion of species by matrix dimension, subset size and number of species with signal added that indicator species analysis reported as having significant associations with groups of negatively co-occurring species. Blue panels represent proportions when signal had been added to the matrices. Red panels are the mean proportions of species for matrices to which no signal had been added. Values of *N* are the number of species with signal added and *Sub* are the size of the species subsets that the null model was run on. White cells in plots indicate no data collected and are associated with matrices with fewer species than required for the subset size.

4.4.2.2 Frequent pattern mining

When no signal was added to the test matrices the maximum number of species associated with significant patterns of negative co-occurrence was 40% (Figure 4-4; N=5, Sub=15). The majority of the matrices with no signal added did not yield enough subsets (> 50) with significant null model results to be able to run the FP-Growth algorithm (Figure 4-4; represented by the white cells in the plots). A subset size of 15 with a total of 20 species in the community resulted in the highest number of species associated with negative co-occurrence when no signal was added to the matrices. As the ratio of subset size to total species number decreased, the number of associations in matrices with no signal added also decreased. For the matrices with added signal the highest match rates were for subset sizes of 15 and total species number of 20. Increasing the number of species with signal (increasing N) results in lower rates of association. Frequent pattern mining produced no associations when the subset size was smaller than the number of species with signal. The majority of matrices did not produce sufficient data to run the FP-Growth algorithm and coincided with values of less than or equal to 0.05 in Figure 4-5 (see Figure G-2 for the limiting similarity results).

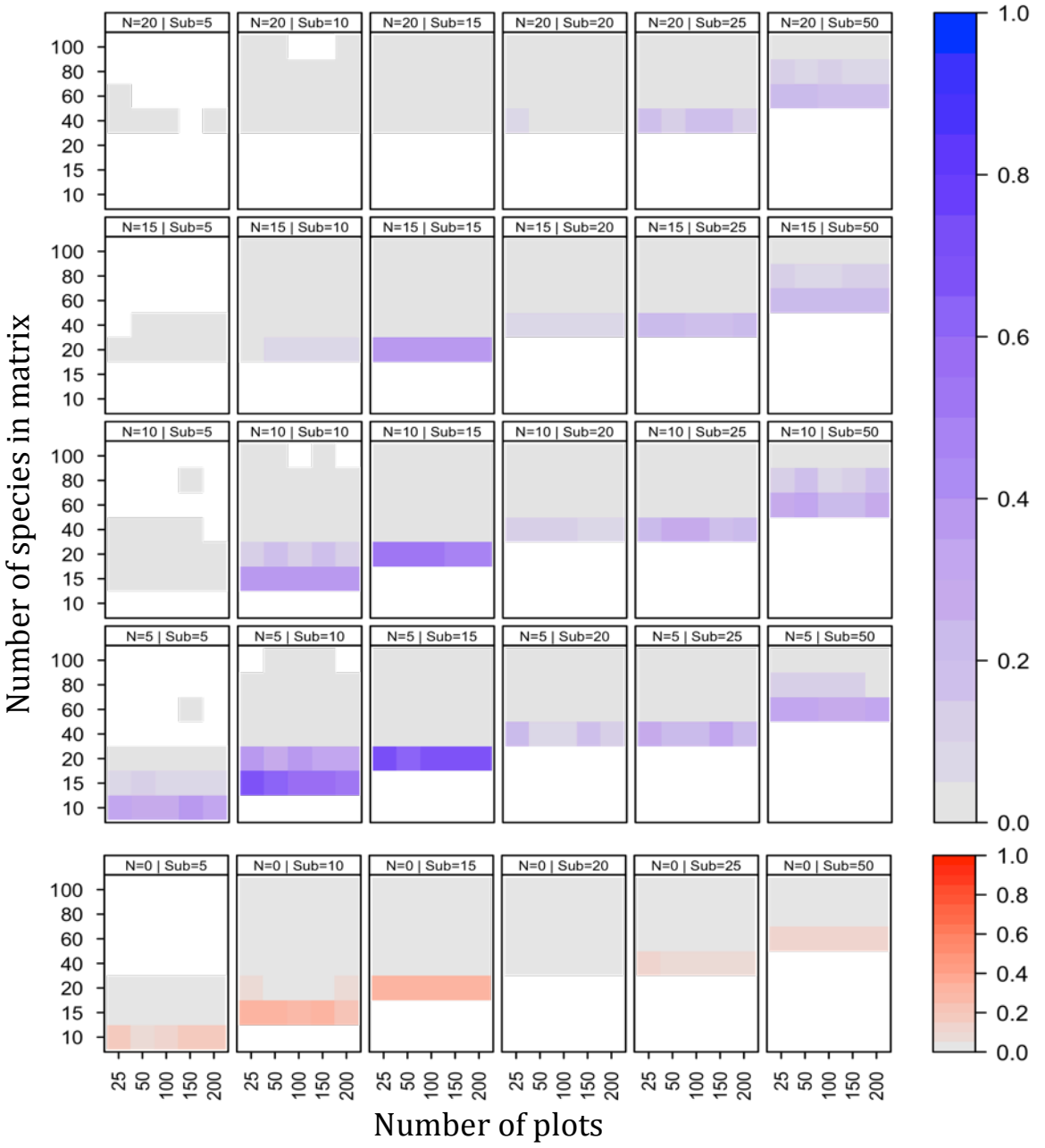

Figure 4-5. The mean proportion of species by matrix dimension, subset size and number of species with signal added that frequent pattern mining indicated as having associations with groups of negatively co-occurring species. Blue panels represent proportions when signal had been added to the matrices. Red panels are the mean proportions of species for matrices to which no signal had been added. Values of *N* are the number of species with signal added and *Sub* are the size of the species subsets that the null model was run on. White cells in plots indicate no data collected and are associated with matrices that have fewer species than required for the subset size. Larger matrices with white cells are due to insufficient data for frequent pattern mining.

4.4.2.3 Probabilistic model of species co-occurrence

The number of species associated with significant patterns of negative co-occurrence increased as either the number of species or the number of plots in the matrices increased (Figure 4-6). This pattern occurred in both the matrices with no added signal and in those with added signal. The largest matrices tested (100 species × 200 plots) resulted in association rates of between 75 and 80% when no signal was added and 75 and 80% when signal was added. The smallest matrices (10 species × 25 plots) resulted in association rates of between zero and five percent when no signal was added and five and ten percent when signal had been added.

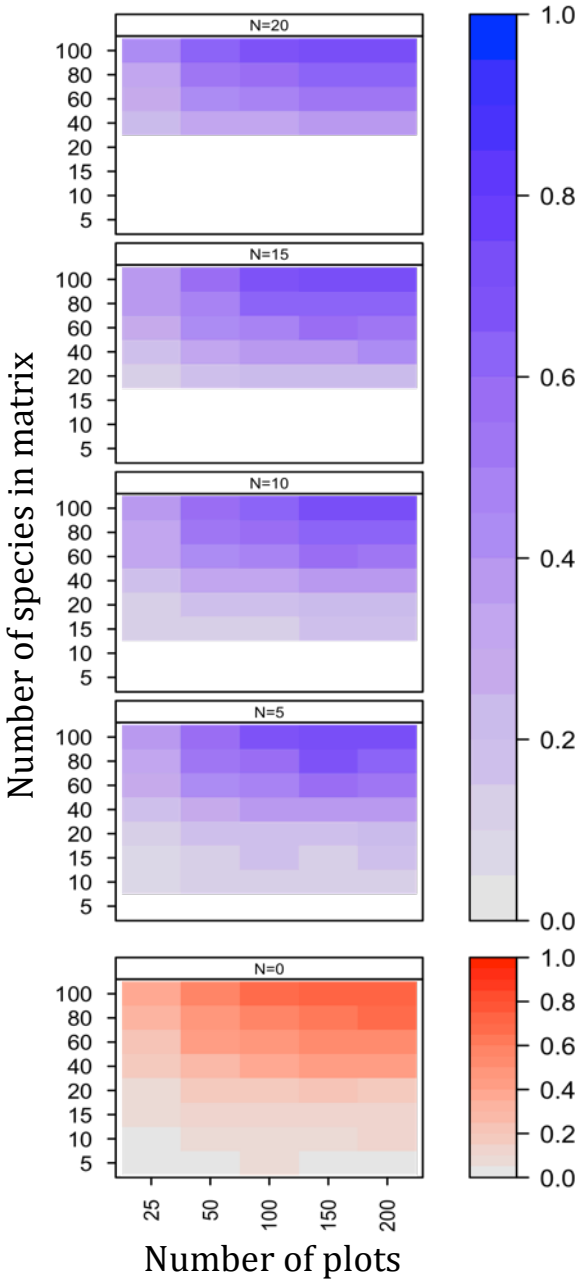

Figure 4-6. The mean proportion of species by matrix dimension, subset size and number of species with signal added that the probabilistic model of co-occurrence indicate as being involved in significant relationships of pairwise negative co-occurrence. Blue panels represent proportions when signal had been added to the matrices. Red panels are the mean proportions of species for matrices to which no signal had been added. Values of *N* are the number of species with signal added. The whole community presence-absence matrices were used for this analysis. White cells in plots indicate no data collected and are associated with matrices with fewer species than required for the subset size.

4.4.2.4 95 percent confidence limit

The results for the 95 percent confidence limit method and the probabilistic model of species co-occurrence resulted in almost identical results for the species verification tests. The proportion of species associated with significant patterns of negative co-occurrence increased as either the number of species or the number of plots in the matrices increased (Figure 4-7). This pattern occurred in both the matrices with no added signal and in those with added signal. The largest matrices tested (100 species × 200 plots) resulted in association rates of between 75 and 80% when no signal was added and 75 and 80% when signal was added. The smallest matrices (10 species × 25 plots) resulted in association rates of between zero and five percent when no signal was added and five and ten percent when signal had been added.

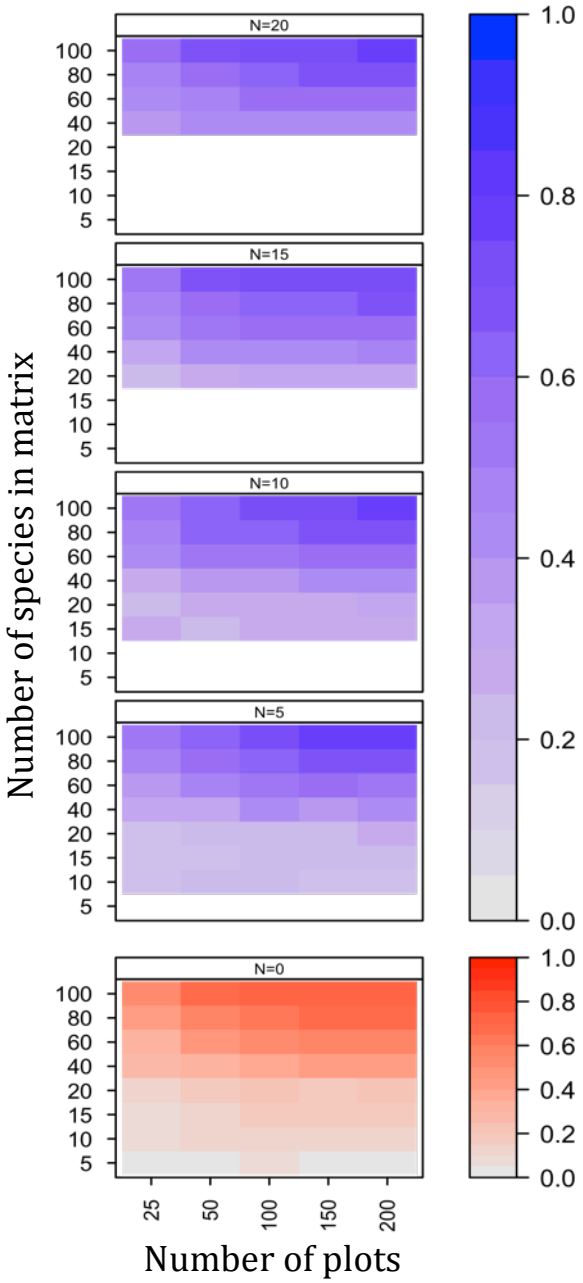

Figure 4-7. The mean proportion of species by matrix dimension, subset size and number of species with signal added that the 95 percent confidence limit criterion indicate as being involved in significant relationships of negative pairwise co-occurrence. Blue panels represent proportions when signal had been added to the matrices. Red panels are the mean proportions of species for matrices to which no signal had been added. Values of *N* are the number of species with signal added. Whole community presence-absence matrices were used for the analyses. White cells in plots indicate no data collected and are associated with matrices with fewer species than required for the subset size.

4.5 Discussion

4.5.1 Species subsetting

The results of our study indicate that limiting similarity and negative co-occurrence null models on subsets of species within a community provides a method to further resolve the details of community assembly. Traditional whole community approaches of null model analyses are able to provide a coarse assessment of community assembly i.e. at the whole community level (Gotelli 2000; Hardy 2008); however, they are unable to do so when community assembly processes occur at the sub-community level (Chapter Three). While pairwise methods of assessing co-occurrence at the sub-community level have been developed (Sfenthourakis *et al.* 2004; 2006; Gotelli & Ulrich 2010; Veech 2013) they still suffer the problem of a trade off between type I and type II statistical errors (Gotelli & Ulrich 2010). No similar methods have been developed for limiting similarity.

While the resolution of our approach does not provide the same level of detail that pairwise tests of co-occurrence or limiting similarity might provide, as it is limited to groups of more than five species (Chapter Two), it does provide a method of gaining greater insights into community assembly using reasonably well understood null model approaches. Although our method of subsetting does not replace the pairwise methods of assessing co-occurrence, it does provide a method of assessing patterns of limiting similarity and negative co-occurrence within groups of species at the sub-community level and may provide a method of determining *n*-way interactions between species that can be used to understand the details of community assembly.

4.5.2 Species identification

One uncertainty of the subsetting method was its ability to identify patterns of either negative co-occurrence or limiting similarity in the correct groups of species where the correct group of species was the group to which signal had been added. Although it was evident that no artefactual pattern was introduced by the subsetting method (see '0sp' plots: Figure 4-1, Figure 4-2, Figure 4-3) a confirmation step was used to try and "recover" the list of species that had been used to add signal to each matrix. None of the four methods

used (frequent pattern analysis, indicator species analysis, the 95 percent confidence limit criterion and the probabilistic model of species co-occurrence) were able to clearly identify the initial list of species used to add signal. This appears, in large part, to be due to each method detecting an overwhelming amount of background pattern in the matrices even when no signal had been added. Because we did not attempt to account for background signal in the test matrices we are not able to conclude that these methods work or even that one method performs better than another; however, while differences between the control matrices (no signal added) and the test matrices are not obvious, they do exist and any future work should attempt to account for background signal. In the case of frequent pattern mining it is possible that the relatively low number of subsets used (1000) per matrix may have resulted in poor frequent pattern detection due to the relatively low numbers of significant subsets included in the analysis. While extending the analysis to use more than 1000 or even hundreds of thousands of subsets is a relatively trivial matter for a single matrix it is computationally problematic for testing 1,000's of matrices. As such a better approach for this type of analysis may be to either increase the number of subsets tested or to test subsets until a total of 1000 significant subsets are obtained. We feel that either of these options is especially important for matrices with larger species numbers due to the fact that as the total number of species in a community increases so do the total number of species subsets. If the number of subsets tested for a given matrix is small it is possible that we have insufficiently sampled from the pool of possible subsets.

4.6 References

Agrawal, R., Imieliński, T. & Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, **22**, 207–216.

Bell, G. (2000). The distribution of abundance in eutral communities. *The American naturalist*, **155**, 606–617.

Cohen, J. (1992). A power primer. *Psychological bulletin*, **112**, 155–159.

Connor, E.F. & Simberloff, D. (1979). The assembly of species communities: Chance or competition? *Ecology*, **60**, 1132–1140.

Cornwell, W.K. & Ackerly, D.D. (2009). Community assembly and shifts in plant trait distributions across an environmental gradient in coastal California. *Ecological Monographs*, **79**, 109–126.

Cornwell, W.K., Schwilk, D.W. & Ackerly, D.D. (2006). A trait-based test for habitat filtering: Convex hull volume. *Ecology*, **87**, 1465–1471.

Dante, S.K., Schamp, B.S. & Aarssen, L.W. (2013). Evidence of deterministic assembly according to flowering time in an old-field plant community. *Functional Ecology*, **27**, 555–564.

de Bello, F., Thuiller, W., Lepš, J., Choler, P., Clément, J.-C., Macek, P., Sebastià, M.-T. & Lavorel, S. (2009). Partitioning of functional diversity reveals the scale and extent of trait convergence and divergence. *Journal of Vegetation Science*, **20**, 475–486.

De Cáceres, M. & Legendre, P. (2009). Associations between species and groups of sites: Indices and statistical inference. *Ecology*, **90**, 3566–3574.

De Cáceres, M., Legendre, P., Wiser, S.K. & Brotons, L. (2012). Using species combinations in indicator value analyses. *Methods in Ecology and Evolution*, **3**, 973–982.

Diamond, J.M. (1975). Assembly of species communties. *Ecology and evolution of communities* pp. 342–444. Belknap Press of Harvard University Press, Cambridge, MA, US.

Dufrêne, M. & Legendre, P. (1997). Species assemblages and indicator species: The need for a flexible asymmetrical approach. *Ecological Monographs*, **67**, 345–366.

Fisher, R.A. & Yates, F. (1953). *Statistical tables for agricultural, biological and medical research*. Edinborough: Oliver & Boyd.

Gause, G.F. (1932). Experimental studies on the struggle for existence I. Mixed population of two species of yeast. *Journal of Experimental Biology*, **9**, 389–402.

Gotelli, N.J. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, **81**, 2606–2621.

Gotelli, N.J. & Ellison, A.M. (2002). Assembly rules for New England ant assemblages. *Oikos*, **99**, 591–599.

Gotelli, N.J. & Rohde, K. (2002). Co-occurrence of ectoparasites of marine fishes: A null model analysis. *Ecology Letters*, **5**, 86–94.

Gotelli, N.J. & Ulrich, W. (2010). The empirical Bayes approach as a tool to identify non-random species associations. *Oecologia*, **162**, 463–477.

Götzenberger, L., de Bello, F., Bråthen, K.A., Davison, J., Dubuis, A., Guisan, A., Lepš, J., Lindborg, R., Moora, M., Pärtel, M., Pellissier, L., Pottier, J., Vittoz, P., Zobel, K. & Zobel, M. (2012). Ecological assembly rules in plant communities--approaches, patterns and prospects. *Biological reviews of the Cambridge Philosophical Society*, **87**, 111–127.

Han, J., Cheng, H., Xin, D. & Yan, X. (2007). Frequent pattern mining: Current status and

future directions. *Data Mining and Knowledge Discovery*, **15**, 55–86.

Han, J., Pei, J. & Yin, Y. (2000). Mining Frequent Patterns Without Candidate Generation. *ACM SIGMOD Record*, **29**, 1–12.

Hardy, O.J. (2008). Testing the spatial phylogenetic structure of local communities: Statistical performances of different null models and test statistics on a locally neutral community. *Journal of Ecology*, **96**, 914–926.

Helsen, K., Hermy, M. & Honnay, O. (2012). Trait but not species convergence during plant community assembly in restored semi-natural grasslands. *Oikos*, EV 1–EV 11.

Hubbell, S.P. (2005). Neutral theory in community ecology and the hypothesis of functional equivalence. *Functional Ecology*, **19**, 166–172.

Hubbell, S.P. (2001). *The unified neutral theory of biodiversity and biogeography*. Princeton University Press.

Kraft, N.J.B. & Ackerly, D.D. (2010). Functional trait and phylogenetic tests of community assembly across spatial scales in an Amazonian forest. *Ecological Monographs*, **80**, 401–422.

Kraft, N.J.B., Valencia, R. & Ackerly, D.D. (2008). Functional traits and niche-based tree community assembly in an Amazonian forest. *Science (New York, N.Y.)*, **322**, 580–582.

Levin, S.A. (1970). Community equilibria and stability, and an extension of the competitive exclusion principle. *American Naturalist*, **104**, 413–423.

MacArthur, R. (1970). Species packing and competitive equilibrium for many species. *Theoretical Population Biology*, **1**, 1–11.

MacArthur, R. & Levins, R. (1967). The limiting similarity, convergence, and divergence of coexisting species. *American Naturalist*, 377–385.

Maestre, F.T. & Reynolds, J.F. (2007). Amount or pattern? Grassland responses to the heterogeneity and availability of two key resources. *Ecology*, **88**, 501–511.

Maltez-Mouro, S., Maestre, F.T. & Freitas, H. (2010). Co-occurrence patterns and abiotic stress in sand-dune communities: Their relationship varies with spatial scale and the stress estimator. *Acta Oecologica*, **36**, 80–84.

McKane, R.B., Johnson, L.C., Shaver, G.R., Nadelhoffer, K.J., Rastetter, E.B., Fry, B., Giblin, A.E., Kielland, K., Kwiatkowski, B.L., Laundre, J.A. & Murray, G. (2002). Resource-based niches provide a basis for plant species diversity and dominance in arctic tundra. *Nature*, **415**, 68–71.

Mouillot, D., Dumay, O. & Tomasini, J.A. (2007). Limiting similarity, niche filtering and functional diversity in coastal lagoon fish communities. *Estuarine, Coastal and Shelf*

Science, **71**, 443–456.

Preston, F.W. (1962a). The Canonical Distribution of Commonness and Rarity: Part I. *Ecology*, **43**, 185–215.

Preston, F.W. (1962b). The Canonical Distribution of Commonness and Rarity: Part II. *Ecology*, **43**, 410–432.

Ribichich, A.M. (2005). From null community to non-randomly structured actual plant assemblages: Parsimony analysis of species co-occurrences. *Ecography*, **28**, 88–98.

Rooney, T.P. (2008). Comparison of co-occurrence structure of temperate forest herb-layer communities in 1949 and 2000. *Acta Oecologica*, **34**, 354–360.

Rosindell, J., Hubbell, S.P. & Etienne, R.S. (2011). The unified neutral theory of biodiversity and biogeography at age ten. *Trends in Ecology \& Evolution*, **26**, 340–348.

Schamp, B.S. & Aarssen, L.W. (2009). The assembly of forest communities according to maximum species height along resource and disturbance gradients. *Oikos*, **118**, 564–572.

Schamp, B.S., Chau, J. & Aarssen, L.W. (2008). Dispersion of traits related to competitive ability in an old-field plant community. *Journal of Ecology*, **96**, 204–212.

Sfenthourakis, S., Giokas, S. & Tzanatos, E. (2004). From sampling stations to archipelagos: Investigating aspects of the assemblage of insular biota. *Global Ecology and Biogeography*, **13**, 23–35.

Sfenthourakis, S., Tzanatos, E. & Giokas, S. (2006). Species co-occurrence: the case of congeneric species and a causal approach to patterns of species association. *Global Ecology and Biogeography*, **15**, 39–49.

Stone, L. & Roberts, A. (1990). The checkerboard score and species distributions. *Oecologia*, **85**, 74–79.

Stubbs, W.J. & Wilson, J.B. (2004). Evidence for limiting similarity in a sand dune community. *Journal of Ecology*, **92**, 557–567.

Ulrich, W. & Gotelli, N.J. (2010). Null model analysis of species associations using abundance data. *Ecology*, **91**, 3384–3397.

Ulrich, W., Ollik, M. & Ugland, K.I. (2010). A meta-analysis of species–abundance distributions. *Oikos*, **119**, 1149–1155.

Veech, J.A. (2013). A probabilistic model for analysing species co-occurrence. *Global Ecology and Biogeography*, **22**, 252–260.

Weiher, E. & Keddy, P.A. (1999). *Ecological assembly rules: perspectives, advances, retreats*.

Cambridge Univ Pr.

Weiher, E., Clarke, G.D.P. & Keddy, P.A. (1998). Community assembly rules, morphological dispersion, and the coexistence of plant species. *Oikos*, **81**, 309–322.

Wilson, J.B. & Stubbs, W.J. (2012). Evidence for assembly rules: Limiting similarity within a saltmarsh. *Journal of Ecology*, **100**, 210–221.

Zhang, J., Hao, Z., Song, B., Li, B., Wang, X. & Ye, J. (2009). Fine-scale species co-occurrence patterns in an old-growth temperate forest. *Forest Ecology and Management*, **257**, 2115–2120.

5.0 General discussion and conclusions

5.1 Synopsis

The works included in this thesis provide a comprehensive statistical analysis of two classes of null model that are commonly used in ecology: limiting similarity, and negative co-occurrence. The purpose of these analyses was to fill current gaps in knowledge and to develop an alternative approach of analysis. The specific goal of developing an alternative analysis is to be able to assess the role of negative co-occurrence and limiting similarity at the sub-community level versus the whole community level. While the concept of community includes all interacting species within a habitat, regardless of taxa or trophic level, the community concept that is most commonly used with these null models is one that restricts species to a single taxonomic grouping. This restricted definition of community is the result of the taxonomic focus of investigators and not necessarily a limitation of the null models. Although the null models were tested under this reduced definition of community, there is no reason to expect them to perform differently with a broader community definition. That is, if the null model randomisation scheme and metric used are applicable across the taxonomic groups, then these null models should perform in accordance with the findings of this thesis.

In Chapter Two, baseline assessments were conducted of the statistical performance for both classes of null model. The range of conditions that were assessed overlapped with those of previous studies (Fayle & Manica, 2010; Gotelli, 2000; Hardy, 2008); however, the range was extended in order to determine if there was any lower threshold to their use. As part of the analyses two randomization algorithms were tested for the limiting similarity null model: abundance weighted trait shuffling, and abundance independent trait shuffling, along with two metrics: mean nearest neighbour trait distance, and the standard deviation of the nearest neighbour trait distance. For the negative co-occurrence analyses the randomization algorithm used was the independent swap (Gotelli, 2000) and the metric used was the C-Score (Stone & Roberts, 1990). As part of theses analyses I also assessed three methods of significance determination: inclusive *p*-values ($p \leq$ or $p \geq$), exclusive *p*-values ($p <$ or $p >$), and SES (standardized effect size). I found that a lower threshold

number of six species, common to all of the null models tested, existed. The only indication of an upper limit existed for the abundance-weighted trait-shuffling algorithm (AWTS), which showed an increase in the type I error rates when the number of plots exceeded 25. It is evident from these results that the use of the AWTS null model is very limited in its application and should be avoided except under very specific conditions. It is also evident from our work that the abundance-independent trait shuffling algorithm (AITS) in combination with either the mean nearest neighbour distances (NN) or standard deviation of the nearest neighbour distances (SDNN) is a reliable tool for assessing limiting similarity within communities. The use of exclusive *p*-values for both the limiting similarity and co-occurrence null models is preferred as exclusive *p*-values reduced the type I error rates without a significant increase in type II error rates.

In Chapter Three, I explore the statistical performance of the negative co-occurrence and limiting similarity models with respect to the proportion of species that contributed signal to patterns of limiting similarity and negative co-occurrence. Synthetic presence-absence matrices were randomly generated covering a broad range of community dimensions. Patterns of limiting similarity and negative co-occurrence were then added the to matrices using known proportions of species and the null models were run for each matrix. The C-Score test statistic and "fixed-fixed" independent swap algorithm were used for co-occurrence null models. For the limiting similarity null models, we used the Nearest Neighbour (NN) trait distance and the Standard Deviation of Nearest Neighbour distances (SDNN) as test statistics, and considered two randomization algorithms: Abundance Independent Trait Shuffling (AITS) and, Abundance Weighted Trait Shuffling (AWTS).

Type II error rates for both null models increased rapidly as the proportion of species contributing signal to co-occurrence or limiting similarity patterns decreased. Acceptable type II error rates ($\beta < 0.30$) were only achieved when 80% or more of species contributed signal to patterns of limiting similarity and when 60% or more of species contributed signal to patterns of negative co-occurrence. While limiting similarity and negative co-occurrence null models perform with acceptable type I and type II error rates when greater than 80% of the community is involved, they significantly under-report patterns when the pattern of interest is driven by a subset of species in the community.

In Chapter Four, I use the knowledge gained from the Chapters Two and Three about the statistical performance of the null models to develop a method that is able to recognize the presence of limiting similarity and negative co-occurrence patterns in communities, even when they are less prominent. Unlike traditional null modelling approaches, I test random subsets of species within matrices, instead of the whole matrices, for pattern. Also, in an effort to identify the groups of species responsible for significant patterns, I test the ability of four methods: indicator species analysis, the probabilistic model of co-occurrence, the 95% confidence limit criterion, and frequent pattern mining, to correctly identify the species used to add signal to the matrices.

Benchmark analyses, using synthetic matrices, indicated that null model analyses on subsets of species perform with acceptable type I and type II error rates and that no artefactual signal was introduced by the subsetting method. The tests used to identify species groups associated with significant subsets were inconclusive due to the significant number of background associations that occur natively in the randomly generated matrices.

5.2 Contributions and significance

The three manuscripts that make up this thesis provide a comprehensive analysis of two commonly used classes of null model in ecology; limiting similarity, and negative co-occurrence. Although previous analyses of these null models (co-occurrence: Gotelli, 2000; Fayle & Manica, 2010; limiting similarity: Hardy, 2008) have provided the groundwork needed to employ them there remained some uncertainty in the range of community dimension that they could be applied to. This uncertainty was resolved in Chapter Two, which also provides recommendations on the use of the two null models. The recommendations being: that exclusive *p*-values should be used for both null models in order to minimize the likelihood of type I statistical errors, both null models should be restricted to matrices with more than five species, the abundance independent trait shuffling (AITS) null model should be used versus the abundance independent trait shuffling null model (AWTS), and the mean nearest trait distance metric (NN) is better choice for limiting similarity null models when compared to the standard deviation of the nearest neighbour distance (SDNN).

Chapter Three adds to our knowledge of the performance of the null models by addressing the statistical power of the tests. While the traditional usage of the null models to assess whole community patterns of limiting similarity and negative co-occurrence are useful they have been unable to provide any insight into the specifics of the community assembly. If only subsets of species in the community contribute to patterns of community assembly, 80% or less, then these null models are unlikely to provide a clear picture of the processes at work.

Chapter Four provides ecologists with a much-needed tool for assessing patterns of limiting similarity and negative co-occurrence in communities when patterns only occur in a subset of the community. While the tool does not yet provide specifics on which species are involved it does allow researches to determine some of the intricacies of the community assembly process.

5.3 Future research

The findings presented in this thesis act as an incremental step towards truly understanding the community assembly process as envisioned by Diamond (1975); however, work still remains before we will be able to disentangle the details of the process. While the subsetting method of assessing patterns of limiting similarity and negative co-occurrence does allow us to test for patterns consistent with community assembly at multiple scales, it is not able to provide detail in terms of which species are involved. While I attempted to address this in Chapter Four using four different measures of association, the results were inconclusive and need further development. One possible approach may be to account for background signal that occurs in the matrices natively. A second approach may be to increase the number of subsets used for the frequent pattern analysis. Since frequent pattern analyses is a tool designed for mining large databases (tens of thousands of transactions) it may be limited in our study by the relatively small number of subsets used (< 1000 subsets). Yet another possible approach may be to increase the range of subset sizes used for each matrix. Because the null models are sensitive to the proportion of species contributing signal and our study was limited to subsets of 20 or fewer species, it is probable that we missed some significant patterns. By increasing the range of subset sizes tested along with the number of subsets tested we may be able increase the amount of

information that the frequent pattern mining has to work with thus increasing it effectiveness at finding groups of species associated with limiting similarity and negative co-occurrence. While these suggestions all represent specific approaches to improving subsetting as a tool there are others, both specific to the subsetting method and to community assembly that can be addressed. For example, is subsetting able to distinguish patterns that occur in more than one set of species. i.e., when multiple groups of species each contribute signal to the same pattern or opposite patterns. Other areas of exploration exist in comparing results between the two classes of null model i.e., are species that negatively co-occur more likely to have similar trait values (trait under-dispersion) or are groups of species with significant trait over-dispersion more likely to neutrally or positively co-occur.

5.4 References

Diamond, J. M. (1975). Assembly of species communities (pp. 342–444). Cambridge, MA, USA: Belknap Press of Harvard University Press.

Fayle, T. M., & Manica, A. (2010). Reducing over-reporting of deterministic co-occurrence patterns in biotic communities. *Ecological Modelling*, *221*(19), 2237–2242.

Gotelli, N. J. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, *81*(9), 2606–2621.

Hardy, O. J. (2008). Testing the spatial phylogenetic structure of local communities: statistical performances of different null models and test statistics on a locally neutral community. *Journal of Ecology*, *96*(5), 914–926.

Stone, L., & Roberts, A. (1990). The checkerboard score and species distributions. *Oecologia*, *85*(1), 74–79.

Appendix A Scala code common to all analyses

A.1 NullModeller library

A.1.1 FactorialMap.scala

```
package ca.mikelavender.nullmodeller

/**
 * This object stores calculated factorials for future use so that they don't
 * have to be recalculated again. This speeds things up enormously!
 */
object FactorialMap {
  // Initialize the map
  private var fact: Map[Int, BigInt] = Map(0 -> 1, 1 -> 1)

  private def build(n: Int) {
    lazy val factsWithPrefix: Stream[BigInt] = BigInt(0) #:: BigInt(1) #:: (factsWithPrefix.zipWithIndex.tail map {
      case (f, i) => f * i
    })

    val facts = factsWithPrefix drop 2

    var counter = 1
    facts take n foreach (f => {
      fact += (counter -> f)
      counter += 1
    })
  }

  def get(n: Int) = {
    if (!fact.contains(n)) build(n)
    if (n <= 1) BigInt(1)
    else fact(n)
  }
}
```

A.1.2 MatrixRandomisation.scala

```
package ca.mikelavender.nullmodeller

import collection.mutable.ArrayBuffer
import concurrent.forkjoin.ThreadLocalRandom
import collection.immutable.Seq

class MatrixRandomisation {
  val helpers = new MatrixRandomisationHelperMethods

  //This method is assuming that the trait array is for only one trait.
```

```
  def traitAWTS(matrixPA: IntMatrix, arrayTrait: Array[Double]): DoubleMatrix = {

    // get the dimensions of the matrix and save them
    val rows = matrixPA.length
    val cols = matrixPA(0).length

    // create a clone of the pa matrix to track if cells are filled or not and to recalculate relative abundance
    val trackingMatrix = matrixPA.map(_.clone())

    // create a zero filled IntMatrix to store our assigned trait indices
    val traitIndicesMatrix = Array.fill(rows, cols)(-1)

    // get the indices for each occupied cell and shuffle them. This will make us populate each
    // cell of the trait matrix in random order.
    val indicesList = shuffle(getIndicesOfPresences(matrixPA))

    // get the current probability matrix
    val weightedTraitValues = getWeightedTraitValues(trackingMatrix)
    for (index <- indicesList) {

      // setup var's to hold trait index and plot tests
      var selectedTrait = 0

      val plotArray = traitIndicesMatrix.map(_(index._2))

      // loop until we have a trait value that is not in the PLOT.
      do {
        selectedTrait = WeightedRandomSelection.weightedSelection(weightedTraitValues.filter(i => i.weight != 0d),
1, Random).head
      } while (!uniqueToPlot(selectedTrait, plotArray))

      // store the index value but increment it by one so that we have a 1 based matrix. 0's are no data
      traitIndicesMatrix(index._1)(index._2) = selectedTrait

      // adjust the tracking matrix
      trackingMatrix(index._1)(index._2) = 0
    }

    // OK... convert the trait indices to trait values
    val shuffledTraitMatrix = Array.fill(rows, cols)(0d)
    for (index <- indicesList) {
      shuffledTraitMatrix(index._1)(index._2) = arrayTrait(traitIndicesMatrix(index._1)(index._2))
    }

    shuffledTraitMatrix
  }

  def uniqueToPlot(traitIndex: Int, plotArray: IntRow): Boolean = !plotArray.contains(traitIndex)

  def getIndicesOfPresences(matrix: IntMatrix): Array[(Int, Int)] = {
    var result = Array[(Int, Int)]()
    for (i <- 0 to matrix.length - 1)
      result = matrix(i).zipWithIndex.filter(p => p._1 != 0).map(f => (i, f._2)) ++ result
```

```
    result
  }

  // get the species abundances wrt the matrix
  def getWeightedTraitValues(matrix: IntMatrix): Seq[WeightedRandomSelection.WeightedItem[Int]] = {
    helpers.getRowProportions(matrix).
      zipWithIndex.
      map(f => WeightedRandomSelection.WeightedItem(f._2, f._1)).
      toSeq
  }

  def traitAITS(matrixPA: IntMatrix, arrayTrait: Array[Double]): DoubleMatrix = {
    val shuffledMatrix = Array.fill(matrixPA.length, matrixPA(0).length)(0d)
    val shuffledTraits: Array[Double] = shuffle(arrayTrait)

    var r = 0
    while (r < matrixPA.length) {
      shuffledMatrix(r) = matrixPA(r).map(_ * shuffledTraits(r))
      r += 1
    }
    shuffledMatrix
  }

  def shuffle[T](arrayTrait: Array[T]): Array[T] = {
    val shuffledTraits = arrayTrait.clone()

    //    To shuffle an array a of n elements (indices 0..n-1):
    for (i <- arrayTrait.length - 1 to 1 by -1) {
      val j = Random.nextInt(i + 1)
      val temp = shuffledTraits(i)
      shuffledTraits(i) = shuffledTraits(j)
      shuffledTraits(j) = temp
    }
    shuffledTraits
  }


  def coOccFixedFixed(matrixPA: IntMatrix, iterations: Int = 30000): IntMatrix = {
    val mtxLength = matrixPA.length //stays constant so don't waste time counting the length of the matrix each
time
    val mtxWidth = matrixPA(0).length //stays constant so don't waste time counting the length of the matrix each
time
    var loopCounter = 0
    while (loopCounter < iterations) {
      if (helpers.findSwaps(matrixPA, mtxLength, mtxWidth))
        loopCounter += 1
    }
    matrixPA
  }

  def coOccFixedEquiProb(matrixPA: IntMatrix): IntMatrix = {
    var counter = 0
    val rowCount = matrixPA.size
```

```
  while (counter < rowCount) {
    val toShuffle = matrixPA(counter)
    matrixPA(counter) = helpers.shuffle(toShuffle)
    counter += 1
  }
  matrixPA
}

def coOccFixedEquiProp(matrixPA: IntMatrix): IntMatrix = {
  /*1. Build prob matrix
  * 2. Populate cells based on prob
  * 3. Find Rows & Columns that do not match constraints
  *   i. if row is over find column that is over with cell turned on
  *   ii. if row is under find column that is under with cell turned off
  *   iii. if that won't correct it then ... just make sure that rows match?*/

  val rows = matrixPA.length
  val cols = matrixPA(0).length
  val cProps = helpers.getColProportions(matrixPA).toIndexedSeq
  val rProps = helpers.getRowProportions(matrixPA).toIndexedSeq
  val filledMatrix = Array.fill(rows, cols)(0)
  val probMatrix = Array.fill(rows, cols)(0d)

  def buildProbMatrix() {
    var row, col = 0
    while (row < rows) {
      while (col < cols) {
        probMatrix(row)(col) = cProps(col)
        col += 1
      }
      row += 1
      col = 0
    }
  }

  def buildPAMatrix() {
    var row, col = 0
    while (row < rows) {
      while (col < cols) {
        filledMatrix(row)(col) = {
          if (ThreadLocalRandom.current().nextDouble() <= probMatrix(row)(col)) 1
          else 0
        }
        col += 1
      }
      row += 1
      col = 0
    }
  }

  def constrainRows() {
    var row = 0
    var workingRow = filledMatrix(row)
```

```
    while (row < rows) {
     workingRow.sum.compare(matrixPA(row).sum) match {
      case -1 => {
       val zeros = helpers.getIndices(workingRow, 0)
       workingRow = workingRow.updated(zeros._1(ThreadLocalRandom.current().nextInt(zeros._2)), 1)
      }
      case 1 => {
       val ones = helpers.getIndices(workingRow, 1)
       workingRow = workingRow.updated(ones._1(ThreadLocalRandom.current().nextInt(ones._2)), 0)
      }
      case 0 => {
       filledMatrix(row) = workingRow.toArray
       row += 1
       if (row < rows) {
        workingRow = filledMatrix(row)
       }
      }
     }

    }
   }

   buildProbMatrix
   buildPAMatrix
   constrainRows

   filledMatrix
  }

  def coOccFixedRange(matrixPA: IntMatrix): IntMatrix = {
   val row = matrixPA.length
   val col = matrixPA(0).length
   var cCounter = 0

   val const = helpers.getFixedRangeConstraints(matrixPA)
   val range = const._1 to const._2
   val rowConst = const._3

   // Create a 0 filled matrix
   val filledMatrix = Array.fill(row, col)(0)

   //Fill the matrix using probabilities from row totals (row Total/# of Cols)
   //build a row
   var rCounter = 0
   //todo: convert this to a while loop
   def buildRow = {
    cCounter = 0
    val rSum = rowConst(rCounter)
    val buf = new ArrayBuffer[Int]()
    while (cCounter < col) {
     buf.append({
      if (ThreadLocalRandom.current().nextDouble() <= rSum / col.toDouble) 1
      else 0
```

```
      })
      cCounter += 1
    }
    buf
  }

  def populateRows() {
    var workingRow = buildRow
    while (rCounter < row) {
      workingRow.sum.compare(rowConst(rCounter)) match {
        case -1 => {
          val zeros = helpers.getIndices(workingRow, 0)
          workingRow = workingRow.updated(zeros._1(ThreadLocalRandom.current().nextInt(zeros._2)), 1)
        }
        case 1 => {
          val ones = helpers.getIndices(workingRow, 1)
          workingRow = workingRow.updated(ones._1(ThreadLocalRandom.current().nextInt(ones._2)), 0)
        }
        case 0 => {
          filledMatrix(rCounter) = workingRow.toArray
          rCounter += 1
          if (rCounter < row) {
            workingRow = buildRow
            populateRows()
          }
        }
      }
    }
  }

  populateRows()
  if (!helpers.checkColConstraints(filledMatrix, range)) {
    for (r <- 0 to row - 1; c <- 0 to col - 1) filledMatrix(r)(c) = 0
    rCounter = 0
    populateRows()
  }
  filledMatrix
 }


}
```

A.1.3 MatrixRandomisationHelperMethods.scala

```
package ca.mikelavender.nullmodeller

 import collection.mutable.ArrayBuffer
 import concurrent.forkjoin.ThreadLocalRandom

 private[nullmodeller] class MatrixRandomisationHelperMethods {
   def findSwaps(matrix: IntMatrix, mtxLength: Int, mtxWidth: Int): Boolean = {
```

```
  val result = findSwapSub(matrix, mtxLength, mtxWidth)

  val zeroOne = getIndices(result._3, -1, mtxWidth)
  val oneZero = getIndices(result._3, 1, mtxWidth)

  val col1 = zeroOne._1(ThreadLocalRandom.current().nextInt(zeroOne._2))
  val col2 = oneZero._1(ThreadLocalRandom.current().nextInt(oneZero._2))

  swap(matrix, result._1, result._2, col1, col2)
  true
}

def findSwapSub(matrix: IntMatrix, mtxLength: Int, mtxWidth: Int): Tuple3[Int, Int, Array[Int]] = {
  var row1, row2 = -1
  var diff = Array(mtxLength)

  do {
    row1 = ThreadLocalRandom.current().nextInt(mtxLength)
    row2 = getNextIndex(row1, mtxLength)
    diff = subRows(matrix(row1), matrix(row2), mtxWidth)
  }
  while (canSwapMethod(diff, mtxWidth) == false)
  (row1, row2, diff)
}

def swap(matrix: IntMatrix, row1: Int, row2: Int, col1: Int, col2: Int) = {

  val temp1 = matrix(row1)(col1)
  matrix(row1)(col1) = matrix(row1)(col2)
  matrix(row1)(col2) = temp1

  val temp2 = matrix(row2)(col1)
  matrix(row2)(col1) = matrix(row2)(col2)
  matrix(row2)(col2) = temp2

  matrix
}

def canSwapMethod(c: Array[Int], w: Int): Boolean = {
  var i = 0
  var min = false
  var max = false
  while (i < w) {
    c(i) match {
      case -1 => min = true
      case 1 => max = true
      case _ => None
    }
    i += 1
  }
  (min & max) == true
}
```

```
def isSwappable(matrix: IntMatrix, mtxLength: Int, mtxWidth: Int): Boolean = {
 var result = false
 var canSwap = 0

 var rCounter = 0
 while (rCounter < mtxLength) {
  var rCounter2 = rCounter + 1
  while (rCounter2 < mtxLength) {
   val difference = subRows(matrix(rCounter), matrix(rCounter2), mtxWidth)

   if (difference.min == -1 & difference.max == 1) {
    canSwap += 1
    result = true
   }
   rCounter2 += 1
  }
  rCounter += 1
 }
 result
}


def getNextIndex(i: Int, constraint: Int): Int = {
 val newIndex = ThreadLocalRandom.current().nextInt(constraint)
 newIndex match {
  case `i` => getNextIndex(i, constraint)
  case _ => newIndex
 }
}

def subRows(a: Array[Int], b: Array[Int], sizeHint: Int): Array[Int] = {
 val l: Array[Int] = new Array(sizeHint)
 var i = 0
 while (i < sizeHint) {
  l(i) = a(i) - b(i)
  i += 1
 }
 l
}

def subRows(a: Array[Double], b: Array[Double], sizeHint: Int): Array[Double] = {
 val l: Array[Double] = new Array(sizeHint)
 var i = 0
 while (i < sizeHint) {
  l(i) = a(i) - b(i)
  i += 1
 }
 l
}

def getIndices(a: ArrayBuffer[Int], v2Index: Int): (Array[Int], Int) = getIndices(a, v2Index, a.length)

def getIndices(a: Array[Int], v2Index: Int): (Array[Int], Int) = getIndices(a, v2Index, a.length)
```

```
def getIndices(a: ArrayBuffer[Int], v2Index: Int, sizeHint: Int): (Array[Int], Int) = {
 var i, cnt = 0
 while (i < sizeHint) {
  if (a(i) == v2Index) {
   cnt += 1
  }
  i += 1
 }
 val l: Array[Int] = new Array(cnt)
 var size = 0 //
 i = 0
 while (i < sizeHint) {
  if (a(i) == v2Index) {
   l(size) = i
   size += 1
  }
  i += 1
 }
 (l, size)
}

def getIndices(a: Array[Int], v2Index: Int, sizeHint: Int): (Array[Int], Int) = {
 var i, cnt = 0
 while (i < sizeHint) {
  if (a(i) == v2Index) {
   cnt += 1
  }
  i += 1
 }
 val l: Array[Int] = new Array(cnt)
 var size = 0 //
 i = 0
 while (i < sizeHint) {
  if (a(i) == v2Index) {
   l(size) = i
   size += 1
  }
  i += 1
 }
 (l, size)
}

def getChoices(a: IndexedSeq[Double], v2Index: Double, sizeHint: Int): Tuple2[IndexedSeq[Int], Int] = {
 val l: ArrayBuffer[Int] = new ArrayBuffer()
 var size = 0 //
 var i = 0
 while (i < sizeHint) {
  if (v2Index <= a(i)) {
   size += 1
   l.append(i)
  }
  i += 1
```

```
  }
  (l, size)
}

def getRowTotals(matrix: IntMatrix): List[Int] = matrix.map(_.sum).toList

def getRowTotals(matrix: DoubleMatrix): List[Double] = matrix.map(_.sum).toList

def getColTotals(matrix: IntMatrix): List[Int] = matrix.transpose.map(_.sum).toList

def getColTotals(matrix: DoubleMatrix): List[Double] = matrix.transpose.map(_.sum).toList

def getMatrixTotal(matrix: IntMatrix): Int = matrix.map(_.sum).sum

def getMatrixTotal(matrix: DoubleMatrix): Double = matrix.map(_.sum).sum

def getRowProportions(matrix: IntMatrix): List[Double] = {
  val mTotal = getMatrixTotal(matrix)
  getRowTotals(matrix).map(_ / mTotal.toDouble)
}

def getRowProportions(matrix: DoubleMatrix): List[Double] = {
  val mTotal = getMatrixTotal(matrix)
  getRowTotals(matrix).map(_ / mTotal)
}

def getColProportions(matrix: IntMatrix): List[Double] = {
  val mTotal = getMatrixTotal(matrix)
  getColTotals(matrix).map(_ / mTotal.toDouble)
}

def getColProportions(matrix: DoubleMatrix): List[Double] = {
  val mTotal = getMatrixTotal(matrix)
  getColTotals(matrix).map(_ / mTotal)
}

def shuffle(array: Array[Int]): Array[Int] = {
  val buf = new ArrayBuffer[Int] ++= array

  def swap(i1: Int, i2: Int) {
    val tmp = buf(i1)
    buf(i1) = buf(i2)
    buf(i2) = tmp
  }

  for (n <- buf.length to 2 by -1) {
    val k = ThreadLocalRandom.current().nextInt(n)
    swap(n - 1, k)
  }
  buf.toArray
}

def checkColConstraints(matrix: IntMatrix, range: Range): Boolean = {
```

```
    var result = true
    val cMatrix = matrix.transpose
    var rCounter = 0
    val rows = cMatrix.length
    val min = range.min
    val max = range.max

    while (rCounter < rows) {
      val rSum = cMatrix(rCounter).sum
      if (rSum < min | rSum > max) result = false
      rCounter += 1
    }
    result
  }

  def getFixedRangeConstraints(matrix: IntMatrix) = {
    val cols = matrix.transpose.map(x => x.sum)
    (cols.sorted.min, cols.sorted.max, matrix.map(x => x.sum))
  }
}
```

A.1.4 MatrixStatistics.scala

```
package ca.mikelavender.nullmodeller

import annotation.tailrec


// Calculates metrics for a whole matrix according to a specific method
class MatrixStatistics {
  val stats = new MatrixStatisticsHelperMethods
  val factorialMap = FactorialMap

  def cScore(myMatrix: Array[Array[Int]], x_row: Int, y_col: Int, normalized: Boolean = false): Double = {
    var checkerboard = 0.0

    var rCounter = 0
    while (rCounter < x_row) {
      var rCounter2 = rCounter + 1
      while (rCounter2 < x_row) {
        checkerboard += {
          normalized match {
            case false => pairwiseCScore(myMatrix(rCounter), myMatrix(rCounter2))
            case true => pairwiseNormalizedCScore(myMatrix(rCounter), myMatrix(rCounter2))
          }
        }
        rCounter2 += 1
      }
      rCounter += 1
    }
    checkerboard / (x_row * (x_row - 1).toDouble / 2.0)
```

```
}

/*
* This is the theoretical maximum C Score for a given matrix. It assumes that all species
* can be arranged in the plots to maximize checkerboardedness which is not realistic. A
* more accurate method would be to shuffle the matrix many times keeping track of the maximum
* C-Score value.
*/
def maxCScore(matrix: Array[Array[Int]]) = {
 var x_row = matrix.size
 var cMax = 0.0

 for (row1 <- 0 to x_row - 1) {
  for (row2 <- row1 + 1 to x_row - 1) {
   var rowA = 0.0
   var rowB = 0.0
   rowA += {
    if (matrix(row1).filter(_ == 1).size > matrix(row2).filter(_ == 0).size) {
     matrix(row2).filter(_ == 0).size
    } else {
     matrix(row1).filter(_ == 1).size
    }
   }
   rowB += {
    if (matrix(row2).filter(_ == 1).size > matrix(row1).filter(_ == 0).size) {
     matrix(row1).filter(_ == 0).size
    } else {
     matrix(row2).filter(_ == 1).size
    }
   }
   cMax += (rowA * rowB / (x_row * (x_row - 1).toDouble / 2.0))
  }
 }
 cMax
}

/*
Stone, L., & Roberts, A. (1990). The checkerboard score and species distributions.
     Oecologia, 85(1), 74–79. doi:10.1007/BF00317345
 */
def pairwiseCScore(i: Array[Int], j: Array[Int]): Double = {
 val sizeHint = i.length
 val l: Array[Int] = new Array(sizeHint)
 var c = 0
 while (c < sizeHint) {
  l(c) = i(c) * j(c)
  c += 1
 }
 val S = l.count(p => p == 1).toDouble
 val ri = i.count(p => p == 1).toDouble
 val rj = j.count(p => p == 1).toDouble

 ((ri - S) * (rj - S))
```

```
  }

  /*
  Gotelli, N. J., & Ulrich, W. (2010). The empirical Bayes approach as a tool to identify non-random species associations.
        Oecologia, 162(2), 463–77. doi:10.1007/s00442-009-1474-y
   */
  def pairwiseNormalizedCScore(i: Array[Int], j: Array[Int]): Double = {
    val sizeHint = i.length
    val l: Array[Int] = new Array(sizeHint)
    var c = 0
    while (c < sizeHint) {
      l(c) = i(c) * j(c)
      c += 1
    }
    val K = l.count(p => p == 1).toDouble
    val ki = i.count(p => p == 1).toDouble
    val kj = j.count(p => p == 1).toDouble

    ((ki - K) * (kj - K)) / (ki * kj)
  }

  /*
 Veech, J. A. (2012). A probabilistic model for analysing species co-occurrence. (P. Peres-Neto, Ed.)
      Global Ecology and Biogeography, n/a–n/a. doi:10.1111/j.1466-8238.2012.00789.x
 */
  def veech(s1: Array[Int], s2: Array[Int]): (Double, Double, Double) = {

    val sizeHint = s1.length
    val l: Array[Int] = new Array(sizeHint)
    var c = 0
    while (c < sizeHint) {
      l(c) = s1(c) * s2(c)
      c += 1
    }

    val N = s1.length //number of sites (plots)

    val N1 = s1.count(p => p == 1) // number of species 1
    val N2 = s2.count(p => p == 1) // number of species 2
    val j = l.count(p => p == 1) // number of co-occurrences of sp1 & sp2

    def exact(i: Int): (Double, Double, Double) = {
      ( {
        for (j <- 0 to i - 1)
        yield
          truncate5((C(N, j) * C(N - j, N2 - j) * C(N - N2, N1 - j)).toDouble / (C(N, N2) * C(N, N1)).toDouble)
      }
        .sum, {
        truncate5((C(N, i) * C(N - i, N2 - i) * C(N - N2, N1 - i)).toDouble / (C(N, N2) * C(N, N1)).toDouble)
      }, {
        for (j <- i + 1 to N)
        yield
```

```
      truncate5((C(N, j) * C(N - j, N2 - j) * C(N - N2, N1 - j)).toDouble / (C(N, N2) * C(N, N1)).toDouble)
    }
     .sum
     )
  }

  exact(j)
}

// returns mean plot level NN and mean plot level SDNTD (SDNN)
def meanAndStdDevNTD(matrix: DoubleMatrix): (Double, Double) = {
  var plotNTD: List[Double] = Nil
  var matrixMeanNTD: List[Double] = Nil
  var matrixStdDevNTD: List[Double] = Nil
  var colList: List[Double] = Nil

  // get a columns worth of data
  for (col <- 0 to matrix(0).length - 1) {
    for (row <- 0 to matrix.length - 1) {
      // get the list of col values
      colList = matrix(row)(col) :: colList
    }

    //remove 0's and sort it
    val sorted = colList.filter(_.compare(0) != 0).sorted
    if (sorted.size == 0) sys.error(" Trying to calculate NN on a column of zeros. ABORT! ABORT! ")

    // reset the colList for the next loop
    colList = Nil

    // calculate the NN between species
    for (i <- 0 to sorted.size - 1) {

      if (i == 0) {
        if (sorted.size <= 1) {
          plotNTD = 0d :: plotNTD
        } else
        // the first one is always between itself and the next one in the list
          plotNTD = sorted(i + 1) - sorted(i) :: plotNTD

      } else if (i == sorted.size - 1) {
        // the last one is always with the previous one in the list
        plotNTD = sorted(i) - sorted(i - 1) :: plotNTD
      }

      else {
        //otherwise check both sides
        val lower = math.abs(sorted(i) - sorted(i - 1))
        val upper = math.abs(sorted(i) - sorted(i + 1))
        plotNTD = {
          if (lower > upper) upper else lower
        } :: plotNTD
      }
```

```
    }
    // plotNTD should contain a list of nearest neighbour distance values one for each species in the list

    matrixMeanNTD = stats.mean(plotNTD) :: matrixMeanNTD
    matrixStdDevNTD = stats.stdDev(plotNTD) :: matrixStdDevNTD
    plotNTD = Nil
   }
   (stats.mean(matrixMeanNTD), stats.mean(matrixStdDevNTD))
  }

  def matrixPerms(matrix: IntMatrix): BigInt = {
   val plots = matrix(0).length
   matrix.foldLeft(BigInt(1))((b, a) => b * factorial(plots) / (factorial(a.sum) * factorial(plots - a.sum)))
  }

  def factorial(n: Int): BigInt = factorialMap.get(n)

  def truncate5(x: Double) = (x * 100000).toInt * 0.00001

  def C(n: Int, r: Int) = factorial(n) / (factorial(r) * factorial(n - r))
}
```

A.1.5 MatrixStatisticsHelperMethods.scala

```
package ca.mikelavender.nullmodeller

import math._


private[nullmodeller] class MatrixStatisticsHelperMethods {

  def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

  def stdDev(numbers: List[Double]): Double = {

    var sum: Double = 0.0
    if (numbers.length >= 2) {
      val avg = mean(numbers)
      val factor: Double = 1.0 / (numbers.length.toDouble - 1)
      for (x: Double <- numbers) {
        sum = sum + ((x - avg) * (x - avg))
      }
      sum = sum * factor
    }
    sqrt(sum)
  }

}
```

A.1.6 nullmodeller.scala

```
package ca.mikelavender

 import util.Random

 package object nullmodeller {
  type IntRow = Array[Int]
  type DoubleRow = Array[Double]
  type IntMatrix = Array[IntRow]
  type DoubleMatrix = Array[DoubleRow]

  val Random = new Random
 }
```

A.1.7 NullModels.scala

```
package ca.mikelavender.nullmodeller

 import scala.actors.Future
 import scala.actors.Futures._

 class NullModels {
  private val metrics = new MatrixStatistics
  private val shuffler = new MatrixRandomisation

  def iSwapNullAnalysisController(matrix: IntMatrix, loops: Int = 5000, throttle: Int = 0, normalized: Boolean =
false): List[Double] = {
    throttle match {
     //     case 0 => threadless(matrix, loops)
     case 1 => threadless(matrix, loops, normalized)
     case _ => threaded(matrix, loops, throttle, normalized)
    }
  }

  private[nullmodeller] def threaded(matrix: IntMatrix, loops: Int, throttle: Int, normalized: Boolean): List[Double]
= {
    val rows = matrix.length
    val cols = matrix(0).length
    val iSwap = new MatrixRandomisation
    val observed = List(metrics.cScore(matrix, rows, cols, normalized))

    var tasks = List[Future[Double]]()
    for (cnt <- 1 to loops) {
     val f = future {
      metrics.cScore(iSwap.coOccFixedFixed(matrix.map(_.clone())), rows, cols, normalized)
     }
     tasks = f :: tasks
     if (throttle == 0) {
      //just do all of them
     } else if (cnt % throttle == 0) f()
```

```
    }
    tasks.map(f => f.apply()) ::: observed
  }

  private[nullmodeller] def threadless(matrix: IntMatrix, loops: Int, normalized: Boolean): List[Double] = {
    val rows = matrix.length
    val cols = matrix(0).length
    val iSwap = new MatrixRandomisation
    var results: List[Double] = List(metrics.cScore(matrix, rows, cols, normalized))

    for (cnt <- 1 to loops) {
      val r = metrics.cScore(iSwap.coOccFixedFixed(matrix.map(_.clone())), rows, cols, normalized)
      results = r :: results
    }
    results
  }

  def nullModelAITS(matrix: Array[Array[Int]], traits: Array[Double], loops: Int = 5000): (List[Double], List[Double])
= {

    val orgtMatrix = Array.ofDim[Double](matrix.length, matrix(0).length)

    for (i <- 0 to matrix.length - 1) orgtMatrix(i) = matrix(i).map(_ * traits(i))

    var meanNTD_AITS: List[Double] = List(metrics.meanAndStdDevNTD(orgtMatrix)._1)
    var meanSDNN_AITS: List[Double] = List(metrics.meanAndStdDevNTD(orgtMatrix)._2)

    var loop = 0
    while (loop < loops) {
      val statsAITS = metrics.meanAndStdDevNTD(shuffler.traitAITS(matrix, traits))
      meanNTD_AITS = statsAITS._1 :: meanNTD_AITS
      meanSDNN_AITS = statsAITS._2 :: meanSDNN_AITS
      loop += 1
    }
    (meanNTD_AITS, meanSDNN_AITS)
  }

  def nullModelAWTS(matrix: Array[Array[Int]], traits: Array[Double], loops: Int = 5000): (List[Double], List[Double])
= {

    val orgtMatrix = Array.ofDim[Double](matrix.length, matrix(0).length)

    for (i <- 0 to matrix.length - 1) orgtMatrix(i) = matrix(i).map(_ * traits(i))

    var meanNTD_AWTS: List[Double] = List(metrics.meanAndStdDevNTD(orgtMatrix)._1)
    var meanSDNN_AWTS: List[Double] = List(metrics.meanAndStdDevNTD(orgtMatrix)._2)

    var loop = 0
    while (loop < loops) {
      val statsAWTS = metrics.meanAndStdDevNTD(shuffler.traitAWTS(matrix, traits))
      meanNTD_AWTS = statsAWTS._1 :: meanNTD_AWTS
      meanSDNN_AWTS = statsAWTS._2 :: meanSDNN_AWTS
      loop += 1
```

```
    }
    (meanNTD_AWTS, meanSDNN_AWTS)
  }

}
```

A.1.8 MatrixRandomisationHelperMethods.scala

```
package ca.mikelavender.nullmodeller

class utils {
  val m1 = new MatrixRandomisationHelperMethods

  def isSwappable(matrix: IntMatrix, mtxLength: Int, mtxWidth: Int): Boolean = {
    m1.isSwappable(matrix: IntMatrix, mtxLength: Int, mtxWidth: Int)
  }

}
```

A.1.9 WeightedRandomSelection.scala

```
package ca.mikelavender.nullmodeller

import util.Random

object WeightedRandomSelection {

  /**
   * Get the number of times an event with probability p occurs in N samples.
   * if R is res, then P(R=n) = p^n q^(N-n) N! / n! / (N-n)!
   * where q = 1-p
   * This has the property that P(R=0) = q^N, and
   * P(R=n+1) = p/q (N-n)/(n+1) P(R=n)
   * Also note that P(R=n+1|R>n) = P(R=n+1)/P(R>n)
   * Uses these facts to work out the probability that the result is zero. If
   * not, then the prob that given that, the result is 1, etc.
   */
  def numEntries(p: Double, N: Int, r: Random): Int = if (p > 0.5) N -
    numEntries(1.0 - p, N, r)
  else if (p < 0.0) 0
  else {
    val q = 1.0 - p
    if (N * p > 100.0) {
      // numeric underflow can be a problem with exact computation, but gaussian approximation is good
      val continuousApprox = r.nextGaussian() * math.sqrt(N * p * q) + N * p
      continuousApprox.round.toInt.max(0).min(N)
    } else {
      // exact approx will not underflow
      var n = 0
      var prstop = math.pow(q, N)
```

```
    var cumulative = 0.0
    while (n < N && (r.nextDouble() * (1 - cumulative)) >= prstop) {
     cumulative += prstop
     prstop *= p * (N - n) / (q * (n + 1))
     n += 1
    }
    n
   }
  }


  case class WeightedItem[T](item: T, weight: Double)

  /**
   * Compute a weighted selection from the given items.
   * cumulativeSum must be the same length as items (or longer), with the ith element containing the sum of all
   * weights from the item i to the end of the list. This is done in a saved way rather than adding up and then
   * subtracting in order to prevent rounding errors from causing a variety of subtle problems.
   */
  private def weightedSelectionWithCumSum[T](items: Seq[WeightedItem[T]], cumulativeSum: List[Double],
numSelections: Int, r: Random): Seq[T] = {
   if (numSelections == 0) Nil
   else {
    val head = items.head
    val nhead = numEntries(head.weight / cumulativeSum.head, numSelections, r)
    List.fill(nhead)(head.item) ++ weightedSelectionWithCumSum(items.tail, cumulativeSum.tail, numSelections -
nhead, r)
   }
  }


  def weightedSelection[T](items: Seq[WeightedItem[T]], numSelections: Int, r: Random): Seq[T] = {
   val cumsum = items.foldRight(List(0.0)) {
    (wi, l) => (wi.weight + l.head) :: l
   }
   weightedSelectionWithCumSum(items, cumsum, numSelections, r)
  }


  def testRandomness[T](items: Seq[WeightedItem[T]], numSelections: Int, r: Random) {
   val runs = 5000
   val indexOfItem = Map.empty ++ items.zipWithIndex.map {
    case (item, ind) => item.item -> ind
   }
   val numItems = items.length
   val bucketSums = new Array[Double](numItems)
   val bucketSumSqs = new Array[Double](numItems)
   for (run <- 0 until runs) {
    // compute chi-squared for a run
    val runresult = weightedSelection(items, numSelections, r)
    val buckets = new Array[Double](numItems)
    for (r <- runresult) buckets(indexOfItem(r)) += 1
    for (i <- 0 until numItems) {
```

```
        val count = buckets(i)
        bucketSums(i) += count
        bucketSumSqs(i) += count * count
      }
    }
    val sumWeights = items.foldLeft(0.0)(_ + _.weight)
    for ((item, ind) <- items.zipWithIndex) {
      val p = item.weight / sumWeights
      val mean = bucketSums(ind) / runs
      val variance = bucketSumSqs(ind) / runs - mean * mean
      val expectedMean = numSelections * p
      val expectedVariance = numSelections * p * (1 - p)
      val expectedErrorInMean = math.sqrt(expectedVariance / runs)
      val text = "Item %10s Mean %.3f Expected %.3f±%.3f Variance %.3f expected %.3f".format(item.item, mean,
expectedMean, expectedErrorInMean, variance, expectedVariance)
      println(text)
    }
  }

  def main(args: Array[String]): Unit = {
    val items = Seq(WeightedItem("Red", 1d / 6), WeightedItem("Blue", 2d / 6), WeightedItem("Green", 3d / 6))
    println(weightedSelection(items, 6, new Random()))
    testRandomness(items, 6, new Random())
  }

}
```

A.1.10 WithoutReplacement.scala

```
package ca.mikelavender.nullmodeller


 object WithoutReplacement {
  def main(args: Array[String]) {
    val runner = new WithoutReplacement

    val items = Seq(
      WeightedItem("v1", 5d / 44),
      WeightedItem("v2", 2d / 44),
      WeightedItem("v3", 5d / 44),
      WeightedItem("v4", 5d / 44),
      WeightedItem("v5", 6d / 44),
      WeightedItem("v6", 5d / 44),
      WeightedItem("v7", 5d / 44),
      WeightedItem("v8", 4d / 44),
      WeightedItem("v9", 5d / 44),
      WeightedItem("v10", 2d / 44)
)

    val result = runner.weightedSelection(items, 6)
    println(result.groupBy(identity).mapValues(_.size))
```

```
  }
 }

 case class WeightedItem[T](item: T, weight: Double)

 class WithoutReplacement {
  val rand = Random

  def weightedSelection[T](items: Seq[WeightedItem[T]], numSelections: Int): Seq[T] = {
   val totalWeight = items.map(_.weight).sum
   println(totalWeight)

   def pick_one: T = {
    var rnd = rand.nextDouble * totalWeight
    println("rnd: " + rnd)
    for (i <- items) {
     if (rnd < i.weight) {
      return i.item
     }
     rnd = rnd - i.weight
    }
    // the above should always return something, but compiler doesn't
    // realise that, hence this:
    return items.head.item
   }

   for (i <- 1 to numSelections) yield pick_one
  }
 }
```

Appendix B Chapter 2.0 Scala Code

B.1 Application Code

B.1.1 BasicTestsRunner.scala

```
package ca.mikelavender.CommunityCompetition.basicMatrixTests

import ca.mikelavender.CommunityCompetition.matrixFactory.MatrixFactory
import ca.mikelavender.nullmodeller.{NullModels, MatrixStatistics}
import java.io.FileWriter
import scala.math._


object BasicTestsRunner {
  def main(args: Array[String]) {
    val runner = new BasicTestsRunner
    (1 to 1000).foreach(i => runner.run)
  }

}

class BasicTestsRunner {

  var loop = 0
  val header = List("id", "sp", "pl", "cSES", "shanE", "simpE", "ra_m").mkString("\t")
  var appendFlag = false
  val outfile = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/cScore x Rank Abundance_Evenness.tsv"

  def run {

    val sp = 50 //util.Random.nextInt(100 - 5) + 5
    val pl = 50 //util.Random.nextInt(100 - 5) + 5

    val matrixFactory = new MatrixFactory
    val matrix = matrixFactory.buildMatrix(sp, pl)

    val evenness = new Evenness
    val rankAbundance = new RankAbundance
    val nm = new NullModels

    val shannonsE = evenness.shannons(matrix)
    val simpsonsE = evenness.simpsons(matrix)

    val rankAbundanceLogTrans: List[(Double, Double)] = rankAbundance.get(matrix).map(f => (math.log10(f._1),
f._2.toDouble + 1d))

    val regressionObject = new LinearRegression(rankAbundanceLogTrans)

    val cNull = nm.iSwapNullAnalysisController(matrix, 1000, 4, normalized = false).reverse
```

```
    val coNullSES = ses(cNull.head, cNull.tail)

    loop += 1
    if (loop < 2) {
      writeToFile(header)
    }
    val row = loop + "\t" + sp + "\t" + pl + "\t" + coNullSES + "\t" + shannonsE + "\t" + simpsonsE + "\t" +
regressionObject.slope

    if (loop % 1 == 0) println(loop)
    writeToFile(row)

  }

  def writeToFile(string: String) {
    val out = new FileWriter(outfile, appendFlag)
    appendFlag = true
    try {
      out.write(string + "\n")
    } finally {
      out.close()
    }
  }

  def ses(obs: Double, exp: List[Double]) = (obs - mean(exp)) / stdDev(exp)

  def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

  def stdDev(numbers: List[Double]): Double = {

    var sum: Double = 0.0
    if (numbers.length >= 2) {
      val avg = mean(numbers)
      val factor: Double = 1.0 / (numbers.length.toDouble - 1)
      for (x: Double <- numbers) {
        sum = sum + (x - avg) * (x - avg)
      }
      sum = sum * factor
    }
    sqrt(sum)
  }
}
```

B.1.2 ChapterOneRunner.scala

```
package ca.mikelavender.chapterone

 import java.util
 import java.text.SimpleDateFormat

object ChapterOneRunner {
```

```
def main(args: Array[String]) {

 val usage =
  """
    | Usage: java -jar chapterone.jar [--species num] [--plots num] [--X num] [--Y num] [--co_iter num]
    |    [--species num] => number of species you would like in the matrix. (default=100)
    |    [--plots num]   => number of plots you would like in the matrix. (default=100)
    |                  can be a list of comma seperated values (no space. eg. 5,10,12)
    |    [--X num]       => number of presence/absence matrices to use. (default=10000)
    |    [--Y num]       => number of matrices used for the trait dispersion null model. (default=5000)
    |    [--co_iter num] => number of null matrices to use for each co-occurrence test. (default=5000)
    |    [--cpu num]     => number of cpu's (threads) to use when running. (default=1)
    |                  the default is to run as a sequential program (one thread/core).
    |                  This setting really only applies on multi-core systems and is intended
    |                  to allow other programs to share computing power of the system.
    |    [--silent]      => suppresses output to the console window.
    |    [--autorun]     => automatically increments the plot size with each complete run starting at 10:
    |                  ** NOTE: overrides --plots values.
    |                    Increments are as follows:
    |                    10 to 25 in increments of 5
    |                    25 to 100 in increments of 25
    |    usage           => prints this message
    |
    |    Examples: java -jar chapterone.jar --species 50 --plots 50 --X 100 --Y 100
    |           java -jar chapterone.jar --species 50 --plots 5,10,20,25,27 --X 100 --Y 100
    |           java -jar chapterone.jar --species 35 --autorun
  """.stripMargin

 val runner = new ChapterOneRunner

 if (args.contains("usage")) {
  println(usage)
  sys.exit(1)
 }
 val argList = args.toList
 type OptionMap = Map[Symbol, Any]

 def nextOption(map: OptionMap, list: List[String]): OptionMap = {
  def isSwitch(s: String) = (s(0) == '-')
  list match {
    case Nil => map
    case "--species" :: value :: tail => nextOption(map ++ Map('species -> value.toInt), tail)
    case "--plots" :: value :: tail => nextOption(map ++ Map('plots -> value.split(",").map(_.toInt).toList), tail)
    case "--X" :: value :: tail => nextOption(map ++ Map('X -> value.toInt), tail)
    case "--Y" :: value :: tail => nextOption(map ++ Map('Y -> value.toInt), tail)
    case "--co_iter" :: value :: tail => nextOption(map ++ Map('co_iter -> value.toInt), tail)
    case "--cpu" :: value :: tail => nextOption(map ++ Map('cpu -> value.toInt), tail)
    case "--autorun" :: tail => nextOption(map ++ Map('autorun -> true), tail)
    case "--silent" :: tail => nextOption(map ++ Map('silent -> true), tail)
    case string :: opt2 :: tail if isSwitch(opt2) => nextOption(map ++ Map('outfile -> string), list.tail)
    case string :: Nil => nextOption(map ++ Map('outfile -> string), list.tail)
    case option :: tail => println("Unknown option " + option)
```

```
      sys.exit(1)
    }
   }
   val options = nextOption(Map(), argList)
   //   println(options)
   runner.start(options)
  }
}

class ChapterOneRunner {
  val utilities = new Utilities
  val metrics = new MatrixStatistics
  val nulls = new NullModels
  val reporter = Reporter
  val mg = new MatrixGenerator
  val shuffler = new MatrixRandomisation


  def start(options: Map[Symbol, Any]) {
   val species = options.getOrElse('species, 100).asInstanceOf[Int]
   var plotCount: List[Int] = Nil
   val X = options.getOrElse('X, 10000).asInstanceOf[Int]
   val Y = options.getOrElse('Y, 5000).asInstanceOf[Int]
   val coIter = options.getOrElse('co_iter, 5000).asInstanceOf[Int]
   val cpu = options.getOrElse('cpu, 1).asInstanceOf[Int]
   val autoRun = options.getOrElse('autorun, false).asInstanceOf[Boolean]
   reporter.silent = options.getOrElse('silent, false).asInstanceOf[Boolean]
   var outfile = ""
   val traitCount = 1

   if (autoRun)
    plotCount = List(3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 50, 75, 100)
   else
    plotCount = options.getOrElse('plots, List(100)).asInstanceOf[scala.List[Int]]

   var x = 0 //setup the p/a loop (X loops)
   println("Starting Runs...")
   for (plots <- plotCount) {
    outfile = getFileName(species, plots, X, Y, coIter, outfile)
    mainLoop(plots)
    x = 0
   }
   println("Run(s) are complete!!!")
   System.exit(0)

   def mainLoop(plots: Int) {

    while (x < X) {
     // setup a list to store the trait Null model results for each matrix
     var meanNTDObserved = 0d
     var meanSDNNObserved = 0d
     var meanNTDExpectedAITS: List[Double] = Nil
     var meanSDNNExpectedAITS: List[Double] = Nil
```

```
var meanNTDExpectedAWTS: List[Double] = Nil
var meanSDNNExpectedAWTS: List[Double] = Nil

// build a random p/a matrix
val paMatrix = mg.buildMatrix(species, plots)

val maxC = metrics.maxCScore(paMatrix)

// this is the slow bit!!!
val paNulls = nulls.iSwapNullAnalysisController(paMatrix, colter, cpu)

var y = 0 // setup the trait randomisation loop (Y loops)
// create the random trait x species values.
val traits = Array.fill(traitCount)(buildTraitArray(species))
while (y < Y) {

 // create an empty trait x p/a matrix
 val tMatrix = Array.fill(species, plots)(0d)
 var sp = 0 // setup the species loop counter
 val workingTraits = {
  if (y != 0) shuffler.traitAITS(traits) else traits
 }

 // copy the trait values to p/a matrix
 while (sp < species) {
  var col = 0 // setup the column loop counter
  while (col < plots) {
   tMatrix(sp)(col) = (
    (workingTraits(traitCount - 1)(sp) *
     paMatrix(sp)(col)) * 100d).toInt / 100d //this makes the trait values 0 to 100 with 2 decimal places
   col += 1
  }
  sp += 1
 }

 y match {
  case 0 => {
   // calculate trait metrics for the un-shuffled traits
   val stats = metrics.meanAndStdDevNTD(tMatrix)
   meanNTDObserved = stats._1
   meanSDNNObserved = stats._2
  }
  case _ => {
   val statsAITS = metrics.meanAndStdDevNTD(tMatrix)
   meanNTDExpectedAITS = statsAITS._1 :: meanNTDExpectedAITS
   meanSDNNExpectedAITS = statsAITS._2 :: meanSDNNExpectedAITS

   val statsAWTS = metrics.meanAndStdDevNTD(shuffler.traitAWTS(paMatrix, traits(0)))
   meanNTDExpectedAWTS = statsAWTS._1 :: meanNTDExpectedAWTS
   meanSDNNExpectedAWTS = statsAWTS._2 :: meanSDNNExpectedAWTS
  }
 }
 y += 1
```

```
      }
      reporter.printNullModelResults(
        paNulls._2, // obsC
        paNulls._3, // expC List
        meanNTDObserved,
        meanSDNNObserved,
        meanNTDExpectedAITS,
        meanSDNNExpectedAITS,
        meanNTDExpectedAWTS,
        meanSDNNExpectedAWTS,
      maxC,
        species, plots, X, Y, coIter, x, outfile)
      x += 1
    }
   }
  }

  def buildTraitArray(species: Int) = Array.fill(species)(math.abs((rnd.nextDouble() + 1) * 100))

  def getFileName(species: Int, plots: Int, X: Int, Y: Int, co_iter: Int, outfile: String) = {
   //   outfile + "_" +
    "sp" + species +
     "p" + plots +
     "X" + X +
     "Y" + Y +
     "i" + co_iter +
     "_" + getTimeStamp +
     ".log"
  }

  def getTimeStamp = {
    val now = util.Calendar.getInstance().getTime
    val formatter = new SimpleDateFormat("yyyyMMddHHmmss")
    formatter.format(now)
  }
}
```

B.1.3 chapteronetests.scala

```
package ca.mikelavender

 import util.Random

 package object chapteronetests {
  type Matrix = Array[Array[Int]]
  val rnd = new Random
  val stats = new Statistics
 }
```

B.1.4 Evenness.scala

```
package ca.mikelavender.CommunityCompetition.basicMatrixTests

/**
 * Both algorithms taken from: http://www.tiem.utk.edu/~gross/bioed/bealsmodules/shannonDI.html
 */

class Evenness {

  /**
   * Calculates Shannon's Evenness (equitability)
   * @param paMatrix an <code>Array[Array[Int]]</code> -> A presence/absence matrix
   * @return <code>double</code> -> Shannon's E
   */
  //todo: test this
  def shannons(paMatrix: Array[Array[Int]]) = {

    val S = paMatrix.length // richness
    val totalOccurrences = paMatrix.foldLeft(0d)((acc, x) => acc + x.sum)
    val ShannonsH = -1d * paMatrix.foldLeft(0d)((acc, x) => acc + (x.sum / totalOccurrences) * math.log(x.sum /
totalOccurrences))

    ShannonsH / math.log(S)
  }

  /**
   * Calculates Simpson's Evenness (equitability)
   * @param paMatrix an <code>Array[Array[Int]]</code> -> A presence/absence matrix
   * @return <code>double</code> -> Simpsons's E
   */
  //todo: test this
  def simpsons(paMatrix: Array[Array[Int]]) = {

    val S = paMatrix.length // richness
    val totalOccurrences = paMatrix.foldLeft(0d)((acc, x) => acc + x.sum)
    val SimpsonsD = 1d / paMatrix.foldLeft(0d)((acc, x) => acc + math.pow(x.sum / totalOccurrences, 2d))

    SimpsonsD / S
  }


}
```

B.1.5 Frequencies.scala

```
package ca.mikelavender.mscthesis.standalonecode

import ca.mikelavender.nullmodeller.MatrixStatistics
import util.Random
```

```
import collection.mutable.ArrayBuffer
import java.io.PrintWriter

object Frequencies {
 def main(args: Array[String]) {
  val runner = new Frequencies
  runner.run
 }
}

class Frequencies {

 def run {
  val mStats = new MatrixStatistics

  val ranges: List[Int] = Range(3, 20, 1).toList //::: List(25, 50, 75, 100)
  val out = new PrintWriter("/Users/MikeLavender/Google Drive/School/MSc Data/frequencies.csv")
  out.print("Hash, Count, Sp, Pl, Dim\n")

  for (sp <- ranges) {
   for (pl <- ranges) {
    var i = 0
    val output = scala.collection.mutable.Map.empty[Int, Int]
    while (i < 10000) {
     val hashValue = buildMatrix(sp, pl).map(_.toList).toList.hashCode()
     if (output.contains(hashValue)) {
      output(hashValue) += 1
     } else output.update(hashValue, 1)

     i += 1
    }
    println("Species: " + sp + " Plots: " + pl)
    try {
     out.print(output.mkString(", " + sp + ", " + pl + ", " + sp * pl + "\n").replace(" -> ", ", "))
     out.print(", " + sp + ", " + pl + ", " + sp * pl + "\n")
    }
//     out.close()
//     System.exit(99)
   }
  }
  out.close()
 }


 private def getNextLogNormal(a: Double) = math.exp(Random.nextGaussian() / (2 * a))

 def getNextLogNormal(maxVal: Int, a: Double = 0.2): Int = {
  if (maxVal <= 0) sys.error("ABORT: Invalid maximum value for log-normal distribution given")
  var l = 0
  while (l == 0) {
   val nextInt = getNextLogNormal(a).toInt
   if (nextInt != 0 & nextInt <= maxVal) l = nextInt
  }
```

```
    l
  }

  def getArrayOfNonZeroLogNormals(species: Int, plots: Int = 0, a: Double = 0.2): Array[Int] = {
    var l: List[Int] = Nil
    while (l.size < species) {
      val nextInt = getNextLogNormal(a).toInt
      if (nextInt != 0 & plots == 0) l = nextInt :: l
      else if (plots > 0 & nextInt <= plots & nextInt != 0) l = nextInt :: l
    }
    l.toArray
  }

  def buildMatrix(species: Int, plots: Int): Array[Array[Int]] = {

    def go: Array[Array[Int]] = {
      val spConstArray = getArrayOfNonZeroLogNormals(species, plots)
      val matrix = Array.fill(species, plots)(0)
      for (spIndex <- 0 to species - 1)
        matrix(spIndex) = buildConstrainedRow(spIndex, plots, spConstArray)
      if (constrainColumns(matrix) & isSwappable(matrix, species, plots)) matrix else go
    }

    go
  }

  def buildConstrainedRow(rowIndex: Int, plots: Int, speciesConstraints: Array[Int]): Array[Int] = {
    var workingRow = Array.fill(plots)(if (Random.nextDouble() <= speciesConstraints(rowIndex).toDouble / plots) 1
else 0)

    def constrainRow {
      workingRow.sum.compare(speciesConstraints(rowIndex)) match {
        case -1 => {
          val zeros = workingRow.zipWithIndex.filter(_._1 == 0).map(_._2)
          workingRow = workingRow.updated(zeros(Random.nextInt(zeros.size)), 1)
          constrainRow
        }
        case 1 => {
          val ones = workingRow.zipWithIndex.filter(_._1 == 1).map(_._2)
          workingRow = workingRow.updated(ones(Random.nextInt(ones.size)), 0)
          constrainRow
        }
        case 0 => // Nothing to do. the row meets constraints
      }
    }
    constrainRow

    workingRow
  }


  // creates a single row for the matrix using column proportion as the constraint
  def buildRow(sites: Int, colProportions: Array[Double]) = {
```

```
  for (c <- 0 to sites - 1)
  yield {
   if (Random.nextDouble() <= colProportions(sites - 1)) 1
   else 0
  }
 }

 // Makes sure that each plot has at least one species in it.
 def constrainColumns(matrix: Array[Array[Int]]): Boolean = {
  if (matrix.
    transpose.
    map(_.sum).
    zipWithIndex.
    filter(_._1 == 0).
    map(_._2).length > 0) false
  else true
 }

 def isSwappable(matrix: Array[Array[Int]], mtxLength: Int, mtxWidth: Int): Boolean = {
  var result = false
  var canSwap = 0

  var rCounter = 0
  while (rCounter < mtxLength) {
   var rCounter2 = rCounter + 1
   while (rCounter2 < mtxLength) {
    val difference = subRows(matrix(rCounter), matrix(rCounter2), mtxWidth)

    if (difference.min == -1 & difference.max == 1) {
     canSwap += 1
     result = true
    }
    rCounter2 += 1
   }
   rCounter += 1
  }
  result
 }

 def subRows(a: Array[Int], b: Array[Int], sizeHint: Int): ArrayBuffer[Int] = {
  val l: ArrayBuffer[Int] = new ArrayBuffer()
  var i = 0
  while (i < sizeHint) {
   l.append(a(i) - b(i))
   i += 1
  }
  l
 }

 val test: Array[Array[Int]] = Array(Array(0, 1), Array(1, 0))
 test.hashCode()
}
```

B.1.6 HelperMethods.scala

```
package ca.mikelavender.chapterone

import collection.mutable.ArrayBuffer
import concurrent.forkjoin.ThreadLocalRandom

class HelperMethods {

  def findSwaps(matrix: Matrix, mtxLength: Int, mtxWidth: Int): Boolean = {

    val result = findSwapSub(matrix, mtxLength, mtxWidth)

    val zeroOne = getIndices(result._3, -1, mtxWidth)
    val oneZero = getIndices(result._3, 1, mtxWidth)

    val col1 = zeroOne._1(ThreadLocalRandom.current().nextInt(zeroOne._2))
    val col2 = oneZero._1(ThreadLocalRandom.current().nextInt(oneZero._2))

    swap(matrix, result._1, result._2, col1, col2)
    true
  }

  def findSwapSub(matrix: Matrix, mtxLength: Int, mtxWidth: Int): Tuple3[Int, Int, Array[Int]] = {
    var row1, row2 = -1
    var diff = Array(mtxLength)

    do {
      row1 = ThreadLocalRandom.current().nextInt(mtxLength)
      row2 = getNextIndex(row1, mtxLength)
      diff = subRows(matrix(row1), matrix(row2), mtxWidth)
    }
    while (canSwapMethod(diff, mtxWidth) == false)
    (row1, row2, diff)
  }

  def canSwapMethod(c: Array[Int], w: Int): Boolean = {
    var i = 0
    var min = false
    var max = false
    while (i < w) {
      c(i) match {
        case -1 => min = true
        case 1 => max = true
        case _ => None
      }
      i += 1
    }
    (min & max) == true
  }
```

```
def isSwappable(matrix: Matrix, mtxLength: Int, mtxWidth: Int): Boolean = {
 var result = false
 var canSwap = 0

 var rCounter = 0
 while (rCounter < mtxLength) {
  var rCounter2 = rCounter + 1
  while (rCounter2 < mtxLength) {
   val difference = subRows(matrix(rCounter), matrix(rCounter2), mtxWidth)

   if (canSwapMethod(difference, mtxWidth)) {
    canSwap += 1
    result = true
   }
   rCounter2 += 1
  }
  rCounter += 1
 }
 result
}

def swap(matrix: Matrix, row1: Int, row2: Int, col1: Int, col2: Int) = {

 val temp1 = matrix(row1)(col1)
 matrix(row1)(col1) = matrix(row1)(col2)
 matrix(row1)(col2) = temp1

 val temp2 = matrix(row2)(col1)
 matrix(row2)(col1) = matrix(row2)(col2)
 matrix(row2)(col2) = temp2

 matrix
}

def getNextIndex(i: Int, constraint: Int): Int = {
 val newIndex = ThreadLocalRandom.current().nextInt(constraint)
 newIndex match {
  case `i` => getNextIndex(i, constraint)
  case _ => newIndex
 }
}

def subRows(a: Array[Int], b: Array[Int], sizeHint: Int): Array[Int] = {
 val l: Array[Int] = new Array(sizeHint)
 var i = 0
 while (i < sizeHint) {
  l(i) = a(i) - b(i)
  i += 1
 }
 l
}
```

```
  def getIndices(a: Array[Int], v2Index: Int, sizeHint: Int): Tuple2[Array[Int], Int] = {
    var i, cnt = 0
    while (i < sizeHint) {
      if (a(i) == v2Index) {
        cnt += 1
      }
      i += 1
    }
    val l: Array[Int] = new Array(cnt)
    var size = 0 //
    i = 0
    while (i < sizeHint) {
      if (a(i) == v2Index) {
        l(size) = i
        size += 1
      }
      i += 1
    }
    (l, size)
  }

  def getChoices(a: IndexedSeq[Double], v2Index: Double, sizeHint: Int): Tuple2[IndexedSeq[Int], Int] = {
    val l: ArrayBuffer[Int] = new ArrayBuffer()
    var size = 0 //
    var i = 0
    while (i < sizeHint) {
      if (v2Index <= a(i)) {
        size += 1
        l.append(i)
      }
      i += 1
    }
    (l, size)
  }

  def getRowTotals(matrix: Matrix): List[Int] = matrix.map(_.sum).toList

  def getRowTotals(matrix: Array[Array[Double]]): List[Double] = matrix.map(_.sum).toList

  def getMatrixTotal(matrix: Matrix): Int = matrix.map(_.sum).sum

  def getMatrixTotal(matrix: Array[Array[Double]]): Double = matrix.map(_.sum).sum

  def getRowProportions(matrix: Matrix): List[Double] = {
    val mTotal = getMatrixTotal(matrix)
    getRowTotals(matrix).map(_ / mTotal.toDouble)
  }

  def getRowProportions(matrix: Array[Array[Double]]): List[Double] = {
    val mTotal = getMatrixTotal(matrix)
    getRowTotals(matrix).map(_ / mTotal)
  }
}
```

B.1.7 Helpers.scala

```
package ca.mikelavender.chapterone

 class Helpers {

  private def getNextLogNormal(a: Double) = math.exp(rnd.nextGaussian() / (2 * a))

  def getNextLogNormal(maxVal: Int, a: Double = 0.2): Int = {
   if (maxVal <= 0) sys.error("ABORT: Invalid maximum value for log-normal distribution given")
   var l = 0
   while (l == 0) {
    val nextInt = getNextLogNormal(a).toInt
    if (nextInt != 0 & nextInt <= maxVal) l = nextInt
   }
   l
  }

  def getArrayOfNonZeroLogNormals(species: Int, plots: Int = 0, a: Double = 0.2): Array[Int] = {
   var l: List[Int] = Nil
   while (l.size < species) {
    val nextInt = getNextLogNormal(a).toInt
    if (nextInt != 0 & plots == 0) l = nextInt :: l
    else if (plots > 0 & nextInt <= plots & nextInt != 0) l = nextInt :: l
   }
   l.toArray
  }
 }
```

B.1.8 LinearRegression.scala

```
package ca.mikelavender.CommunityCompetition.basicMatrixTests

/**
 * Source:
 * http://eric-mariacher.blogspot.ca/2011/12/here-is-how-to-compute-simple-linear.html
 *
 * This is a scala port of the java code here:
 * http://introcs.cs.princeton.edu/java/97data/LinearRegression.java.html
 *
 * below are the original comments for the Java code
 */

/******************************************************************************
  *  Compilation:  javac LinearRegression.java StdIn.java
  *  Execution:    java LinearRegression < data.txt
  *
  *  Reads in a sequence of pairs of real numbers and computes the
  *  best fit (least squares) line y  = ax + b through the set of points.
```

```
  *  Also computes the correlation coefficient and the standard error
  *  of the regression coefficients.
  *
  *  Note: the two-pass formula is preferred for stability.
  *
  ************************************************************************/

//todo: test this
 class LinearRegression(pairs: List[(Double, Double)]) {
  private val size = pairs.size

  // first pass: read in data, compute xbar and ybar
  private val sums = pairs.foldLeft(new X_X2_Y(0D, 0D, 0D))(_ + new X_X2_Y(_))
  private val bars = (sums.x / size, sums.y / size)

  // second pass: compute summary statistics
  private val sumstats = pairs.foldLeft(new X2_Y2_XY(0D, 0D, 0D))(_ + new X2_Y2_XY(_, bars))

  private val beta1 = sumstats.xy / sumstats.x2
  private val beta0 = bars._2 - (beta1 * bars._1)
  private val betas = (beta0, beta1)

//  println("y = " + ("%4.3f" format beta1) + " * x + " + ("%4.3f" format beta0))

  // analyze results
  private val correlation = pairs.foldLeft(new RSS_SSR(0D, 0D))(_ + RSS_SSR.build(_, bars, betas))
  private val R2 = correlation.ssr / sumstats.y2
  private val svar = correlation.rss / (size - 2)
  private val svar1 = svar / sumstats.x2
  private val svar0 = (svar / size) + (bars._1 * bars._1 * svar1)
  private val svar0bis = svar * sums.x2 / (size * sumstats.x2)

  def slope = beta1
}

object RSS_SSR {
  def build(p: (Double, Double), bars: (Double, Double), betas: (Double, Double)): RSS_SSR = {
    val fit = (betas._2 * p._1) + betas._1
    val rss = (fit - p._2) * (fit - p._2)
    val ssr = (fit - bars._2) * (fit - bars._2)
    new RSS_SSR(rss, ssr)
  }
}

class RSS_SSR(val rss: Double, val ssr: Double) {
  def +(p: RSS_SSR): RSS_SSR = new RSS_SSR(rss + p.rss, ssr + p.ssr)
}

class X_X2_Y(val x: Double, val x2: Double, val y: Double) {
  def this(p: (Double, Double)) = this(p._1, p._1 * p._1, p._2)

  def +(p: X_X2_Y): X_X2_Y = new X_X2_Y(x + p.x, x2 + p.x2, y + p.y)
}
```

```
class X2_Y2_XY(val x2: Double, val y2: Double, val xy: Double) {
  def this(p: (Double, Double), bars: (Double, Double)) = this((p._1 - bars._1) * (p._1 - bars._1), (p._2 - bars._2) *
(p._2 - bars._2), (p._1 - bars._1) * (p._2 - bars._2))

  def +(p: X2_Y2_XY): X2_Y2_XY = new X2_Y2_XY(x2 + p.x2, y2 + p.y2, xy + p.xy)
 }
```

B.1.9 MatrixFiller.scala

```
package ca.mikelavender.chapterone

// Creates matrices according the specific constraints and distributions
class MatrixFiller {
  val utils = new Utilities


  def fill(matrix: Matrix) = {
    val dims = getDimensions(matrix)
    val row = dims._1
    val col = dims._2

    val const = getFixedRangeConstraints(matrix)
    val range = const._1 to const._2
    val rowConst = const._3

    //Fill the matrix using probabilities from row totals (row Total/# of Cols)
    val filledMatrix = createZeroFilledMatrix((row, col))

    //build a row
    var rCounter = 0
    def buildRow = {
      val rSum = rowConst(rCounter)
      for (c <- 0 to col - 1)
      yield {
        if (rnd.nextDouble() <= rSum / col.toDouble) 1
        else 0
      }
    }

    def populateRows() {
      while (rCounter < row) {
        var workingRow = buildRow
        workingRow.sum.compare(rowConst(rCounter)) match {
          case -1 => {
            val zeros = workingRow.zipWithIndex.filter(_._1 == 0).map(_._2)
            workingRow = workingRow.updated(zeros(rnd.nextInt(zeros.size)), 1)
            populateRows()
          }
          case 1 => {
```

```
          val ones = workingRow.zipWithIndex.filter(_._1 == 1).map(_._2)
          workingRow = workingRow.updated(ones(rnd.nextInt(ones.size)), 0)
          populateRows()
        }
        case 0 => {
          filledMatrix(rCounter) = workingRow.toArray
          rCounter += 1
          populateRows()
        }
      }
    }
  }

  populateRows()
  //   val timer = utils.optTimer(populateRows())
  if (!checkConstraints(filledMatrix, rowConst, range)) {
    for (r <- 0 to row - 1; c <- 0 to col - 1) filledMatrix(r)(c) = 0
    rCounter = 0
    populateRows()
  }
  //   println(timer)

  filledMatrix
}

def getDimensions(matrix: Matrix): (Int, Int) = (matrix.size, matrix(0).size)

def getFixedRangeConstraints(matrix: Matrix) = {
  val cols = matrix.transpose.zipWithIndex.map(x => x._1.sum)
  (cols.sorted.min, cols.sorted.max, matrix.zipWithIndex.map(x => x._1.sum))
}

def createZeroFilledMatrix(dim: Tuple2[Int, Int]): Matrix = (Array.fill(dim._1, dim._2)(0))

def checkConstraints(matrix: Matrix, rowConst: Array[Int], range: Range): Boolean = {
  val transMatrix = matrix.transpose.zipWithIndex.filter(x => !range.contains(x._1.sum)).map(_._2)
  if (transMatrix.size > 0) false else true
}

def textFill() {
  for (loop <- 0 to 1000) {
    var row = ""
    for (r <- 0 to 200) {
      for (c <- 0 to 400) {
        row = row + (if (rnd.nextDouble() <= rnd.nextInt(320) / 400d) "1 " else "0 ")
      }
      //      println(row)
      row = ""
    }
  }
}
}
```

B.1.10 MatrixGenerator.scala

```
package ca.mikelavender.chapterone

 class MatrixGenerator {
  private val helper = new Helpers
  private val helper2 = new HelperMethods

  /*Builds a random matrix using row and column constraints.
  * Row constraints - abundance for each row sampled from log-normal distribution
  * Column constraints - proportion for each column sampled from uniform distribution
  * For methods see:
  * Ulrich, W., & Gotelli, N. J. (2010). Null model analysis of species associations using abundance data. Ecology,
91(11), 3384–3397. doi:10.1890/09-2157.1
  * Gotelli, N. J. (2000). Null Model Analysis of Species Co-Occurrence Patterns. Ecology, 81(9), 2606.
doi:10.2307/177478
  * */

  //todo: I don't need to constrain the random matrix to proportion only the null matrices which are constrained by
the original random matrix proportions.
  def buildMatrix(species: Int, plots: Int): Matrix = buildMatrix(species, plots, "logNormal")

  def buildMatrix(species: Int, plots: Int, speciesConstraint: String): Matrix = {

    def go: Matrix = {
      val spConstArray = getSpeciesConstraints(species, plots, speciesConstraint)
      val matrix = Array.fill(species, plots)(0)
      for (spIndex <- 0 to species - 1)
        matrix(spIndex) = buildConstrainedRow(spIndex, plots, spConstArray)
      if (constrainColumns(matrix) & helper2.isSwappable(matrix, species, plots)) matrix else go
    }

    go
  }

  def buildConstrainedRow(rowIndex: Int, plots: Int, speciesConstraints: Array[Int]): Array[Int] = {
    var workingRow = Array.fill(plots)(if (rnd.nextDouble() <= speciesConstraints(rowIndex).toDouble / plots) 1 else
0)

    def constrainRow {
      workingRow.sum.compare(speciesConstraints(rowIndex)) match {
        case -1 => {
          val zeros = workingRow.zipWithIndex.filter(_._1 == 0).map(_._2)
          workingRow = workingRow.updated(zeros(rnd.nextInt(zeros.size)), 1)
          constrainRow
        }
        case 1 => {
          val ones = workingRow.zipWithIndex.filter(_._1 == 1).map(_._2)
          workingRow = workingRow.updated(ones(rnd.nextInt(ones.size)), 0)
          constrainRow
        }
        case 0 => // Nothing to do. the row meets constraints
      }
```

```
    }
    constrainRow

    workingRow
  }


  // creates a single row for the matrix using column proportion as the constraint
  def buildRow(sites: Int, colProportions: Array[Double]) = {
    for (c <- 0 to sites - 1)
    yield {
      if (rnd.nextDouble() <= colProportions(sites - 1)) 1
      else 0
    }
  }

  // Makes sure that each plot has at least one species in it.
  def constrainColumns(matrix: Matrix): Boolean = {
    if (matrix.
      transpose.
      map(_.sum).
      zipWithIndex.
      filter(_._1 == 0).
      map(_._2).length > 0) false
    else true
  }

  /*
   * Checks that the column constraints are still met after we have adjusted for the row constraints.
   * Allows for the column proportion to be out by 0.0001
   */
  @deprecated
  def checkConstraints(matrix: Matrix, colProps: Array[Double]): Boolean = {
    val mSum = matrix.map(_.sum).sum
    val transMatrix = matrix.transpose.zipWithIndex.filter(x => (math.abs((x._1.sum / mSum.toDouble) -
colProps(x._2)).compareTo(0.0001) == 1)).map(_._2)
    if (transMatrix.size > 0) false else true
  }

  def getSpeciesConstraints(species: Int, plots: Int, speciesConstraint: String): Array[Int] = {
    speciesConstraint match {
      case "logNormal" => helper.getArrayOfNonZeroLogNormals(species, plots)
      //todo: need to implement these in helpers class
      case "normal" => sys.error("Normal distribution not implemented yet")
      case "uniform" => sys.error("Uniform distribution not implemented yet")
      case _ => sys.error("ABORT: No valid distribution given for species abundances")
    }
  }
}
```

B.1.11 MatrixRandomisation.scala

```
package ca.mikelavender.chapterone

 import concurrent.forkjoin.ThreadLocalRandom
 import util.Random
 import collection.immutable.Seq

 class MatrixRandomisation {
   val helpers = new HelperMethods
   val util = new Utilities

  //This method is assuming that the trait array is for only one trait.
   def traitAWTS(matrixPA: Matrix, arrayTrait: Array[Double]): Array[Array[Double]] = {

    // get the dimensions of the matrix and save them
    val rows = matrixPA.length
    val cols = matrixPA(0).length

    // create a clone of the pa matrix to track if cells are filled or not and to recalculate relative abundance
    val trackingMatrix = matrixPA.map(_.clone())

    // create a zero filled IntMatrix to store our assigned trait indices
    val traitIndicesMatrix = Array.fill(rows, cols)(-1)

    // get the indices for each occupied cell and shuffle them. This will make us populate each
    // cell of the trait matrix in random order.
    val indicesList = shuffle(getIndicesOfPresences(matrixPA))

    // get the current probability matrix
    val weightedTraitValues = getWeightedTraitValues(trackingMatrix)
    for (index <- indicesList) {

     // setup var's to hold trait index and plot tests
     var selectedTrait = 0

     val plotArray = traitIndicesMatrix.map(_(index._2))

     // loop until we have a trait value that is not in the PLOT.
     do {
       selectedTrait = WeightedRandomSelection.weightedSelection(weightedTraitValues.filter(i => i.weight != 0d),
1, Random).head
     } while (!uniqueToPlot(selectedTrait, plotArray))

     // store the index value but increment it by one so that we have a 1 based matrix. 0's are no data
     traitIndicesMatrix(index._1)(index._2) = selectedTrait

     // adjust the tracking matrix
     trackingMatrix(index._1)(index._2) = 0
    }

    // OK... convert the trait indices to trait values
    val shuffledTraitMatrix = Array.fill(rows, cols)(0d)
```

```
  for (index <- indicesList) {
    shuffledTraitMatrix(index._1)(index._2) = arrayTrait(traitIndicesMatrix(index._1)(index._2))
  }

  shuffledTraitMatrix
}

def uniqueToPlot(traitIndex: Int, plotArray: Array[Int]): Boolean = !plotArray.contains(traitIndex)

def getIndicesOfPresences(matrix: Matrix): Array[(Int, Int)] = {
  var result = Array[(Int, Int)]()
  for (i <- 0 to matrix.length - 1)
    result = matrix(i).zipWithIndex.filter(p => p._1 != 0).map(f => (i, f._2)) ++ result
  result
}

// get the species abundances wrt the matrix
def getWeightedTraitValues(matrix: Matrix): Seq[WeightedRandomSelection.WeightedItem[Int]] = {
  helpers.getRowProportions(matrix).
    zipWithIndex.
    map(f => WeightedRandomSelection.WeightedItem(f._2, f._1)).
    toSeq
}

def shuffle[T](arrayTrait: Array[T]): Array[T] = {
  val shuffledTraits = arrayTrait.clone()

  //    To shuffle an array a of n elements (indices 0..n-1):
  for (i <- arrayTrait.length - 1 to 1 by -1) {
    val j = Random.nextInt(i + 1)
    val temp = shuffledTraits(i)
    shuffledTraits(i) = shuffledTraits(j)
    shuffledTraits(j) = temp
  }
  shuffledTraits
}

def traitAITS(matrixTrait: Array[Array[Double]]): Array[Array[Double]] = {
  val shuffledTraitMatrix = matrixTrait.map(_.clone())
  val cols = matrixTrait(0).length
  var shuffles = 0
  while (shuffles < cols) {
    val col1 = ThreadLocalRandom.current().nextInt(cols)
    val col2 = ThreadLocalRandom.current().nextInt(cols)
    for (row <- shuffledTraitMatrix) {
      val temp = row(col1)
      row(col1) = row(col2)
      row(col2) = temp
      shuffles += 1
    }
  }
  shuffledTraitMatrix
}
```

```
  def coOccFixedFixed(matrixPA: Array[Array[Int]], iterations: Int = 30000): Matrix = {
    val mtxLength = matrixPA.length //stays constant so don't waste time counting the length of the matrix each
time
    val mtxWidth = matrixPA(0).length //stays constant so don't waste time counting the length of the matrix each
time
    var loopCounter, failCounter = 0
    //    val iterations = mtxLength * mtxWidth
    while (loopCounter < iterations) {
      if (helpers.findSwaps(matrixPA, mtxLength, mtxWidth)) {
        loopCounter += 1
      }
      //      else failCounter += 1
    }
    //    println("Swaps: " + (loopCounter - failCounter) + "\tFailed swaps: " + failCounter)
    matrixPA
  }

  def coOccFixedEquiProb(matrixPA: Array[Array[Int]]) {}

  def coOccFixedEquiProp(matrixPA: Array[Array[Int]]) {}

  def coOccFixedRange(matrixPA: Array[Array[Int]]) {}


}
```

B.1.12 MatrixStatistics.scala

```
package ca.mikelavender.chapterone


// Calculates metrics for a whole matrix according to a specific method
class MatrixStatistics {

  def cScore(myMatrix: Array[Array[Int]]): Double = {
    val x_row = myMatrix.size
    val y_col = myMatrix(0).length
    cScore(myMatrix, x_row, y_col)
  }

  def cScore(myMatrix: Array[Array[Int]], x_row: Int, y_col: Int): Double = {
    val x_row = myMatrix.size
    val y_col = myMatrix(0).length
    var checkerboard = 0.0

    var rCounter = 0
    while (rCounter < x_row) {
      var rCounter2 = rCounter + 1
      while (rCounter2 < x_row) {
        var rowA = 0.0
```

```
    var rowB = 0.0
    var cCounter = 0
    while (cCounter < y_col) {
     myMatrix(rCounter)(cCounter) match {
      case 1 => if (myMatrix(rCounter2)(cCounter) == 0) rowA += 1.0
      case 0 => if (myMatrix(rCounter2)(cCounter) == 1) rowB += 1.0
     }
     cCounter += 1
    }
    checkerboard += (rowA * rowB / (x_row * (x_row - 1).toDouble / 2.0))
    rCounter2 += 1
   }
   rCounter += 1
  }
  checkerboard
}

def maxCScore(matrix: Array[Array[Int]]) = {
  var x_row = matrix.size
  var cMax = 0.0

  for (row1 <- 0 to x_row - 1) {
   for (row2 <- row1 + 1 to x_row - 1) {
    var rowA = 0.0
    var rowB = 0.0
    rowA += {
     if (matrix(row1).filter(_ == 1).size > matrix(row2).filter(_ == 0).size) {
      matrix(row2).filter(_ == 0).size
     } else {
      matrix(row1).filter(_ == 1).size
     }
    }
    rowB += {
     if (matrix(row2).filter(_ == 1).size > matrix(row1).filter(_ == 0).size) {
      matrix(row1).filter(_ == 0).size
     } else {
      matrix(row2).filter(_ == 1).size
     }
    }
    cMax += (rowA * rowB / (x_row * (x_row - 1).toDouble / 2.0))
   }
  }
  cMax
}

// Returns meanNTD (mean NN) and meanSDNN. Both are means of the plot level values.
def meanAndStdDevNTD(matrix: Array[Array[Double]]): (Double, Double) = {
  var plotNTD: List[Double] = Nil
  var matrixMeanNTD: List[Double] = Nil
  var matrixStdDevNTD: List[Double] = Nil
  var colList: List[Double] = Nil

  // get a columns worth of data
```

```
    for (col <- 0 to matrix(0).length - 1) {
     for (row <- 0 to matrix.length - 1) {
      // get the list of col values
      colList = matrix(row)(col) :: colList
     }

     //remove 0's and sort it
     val sorted = colList.filter(_.compare(0) != 0).sorted
     if (sorted.size == 0) sys.error("ABORT! ABORT! " + sorted)

     // reset the colList for the next loop
     colList = Nil

     // calculate the NN between species
     for (i <- 0 to sorted.size - 1) {

      if (i == 0) {
       if (sorted.size <= 1) {
        plotNTD = 0 :: plotNTD
       } else
       // the first one is always between itself and the next one in the list
        plotNTD = sorted(i + 1) - sorted(i) :: plotNTD

      } else if (i == sorted.size - 1) {
       // the last one is always with the previous one in the list
       plotNTD = sorted(i) - sorted(i - 1) :: plotNTD
      }

      else {
       //otherwise check both sides
       val lower = math.abs(sorted(i) - sorted(i - 1))
       val upper = math.abs(sorted(i) - sorted(i + 1))
       plotNTD = {
        if (lower > upper) upper else lower
       } :: plotNTD
      }
     }
     // plotNTD should contain a list of nearest neighbour distance values one for each species in the list

     matrixMeanNTD = stats.mean(plotNTD) :: matrixMeanNTD
     matrixStdDevNTD = stats.stdDev(plotNTD) :: matrixStdDevNTD
     plotNTD = Nil
    }
    (stats.mean(matrixMeanNTD), stats.mean(matrixStdDevNTD))
   }
  }
```

B.1.13 NullModels.scala

```
package ca.mikelavender.chapterone
```

```
import scala.actors.Future
import scala.actors.Futures._

class NullModels {
  private val metrics = new MatrixStatistics

  def iSwapNullAnalysisController(matrix: Matrix, iter: Int, throttle: Int): (String, Double, List[Double]) = {
    throttle match {
//      case 0 => threadless(matrix, iter)
      case 1 => threadless(matrix, iter)
      case _ => threaded(matrix,iter, throttle)
    }
  }


  def threaded(matrix: Matrix, iter: Int, throttle: Int): (String, Double, List[Double]) = {
    val rows = matrix.length
    val cols = matrix(0).length
    val iSwap = new MatrixRandomisation
    val observedResult = metrics.cScore(matrix, rows, cols)

    var tasks = List[Future[Double]]()
    for (cnt <- 1 to iter) {
      val f = future {
        metrics.cScore(iSwap.coOccFixedFixed(matrix.map(_.clone())), rows, cols)
      }
      tasks = f :: tasks
      if (throttle != 0 & cnt % throttle == 0) f()
    }
    val expectedResult = tasks map (f => f.apply())
    ("iswap", observedResult, expectedResult)
  }

  def threadless(matrix: Matrix, iter: Int): (String, Double, List[Double]) = {
    val rows = matrix.length
    val cols = matrix(0).length
    val iSwap = new MatrixRandomisation
    val observedResult = metrics.cScore(matrix, rows, cols)

    var expectedResult = List[Double]()
    for (cnt <- 1 to iter) {
      val r = metrics.cScore(iSwap.coOccFixedFixed(matrix.map(_.clone())), rows, cols)
      expectedResult = r :: expectedResult
    }
    ("iswap", observedResult, expectedResult)
  }
}
```

B.1.14 RankAbundance.scala

```
package ca.mikelavender.CommunityCompetition.basicMatrixTests
```

```
/**
 * Created with IntelliJ IDEA.
 * User: MikeLavender
 * Date: 1/18/2014
 * Time: 4:08 PM
 */
class RankAbundance {

  //todo: test this
  def get(paMatrix: Array[Array[Int]]): List[(Int, Int)] = {
    {
      for (x <- paMatrix ) yield {
        (x.sum)
      }
    }.sorted.reverse.zipWithIndex.toList
  }

}
```

B.1.15 Reporter.scala

```
package ca.mikelavender.chapterone

import java.io.FileWriter
import java.text.DecimalFormat

object Reporter {
  var appendFlag = false
  var header = false
  var currentFile = ""
  var silent = false

  val formatter = new DecimalFormat("#0.00")
  val pValue = new DecimalFormat("#0.0000")
  val stat = new Statistics

  def writeToFile(string: String) {
    writeToFile("results/runlog.csv", string)
  }

  def writeToFile(file: String, string: String) {
    val out = new FileWriter(file, appendFlag)
    appendFlag = true
    try {
      out.write(string + "\n")
    } finally {
      out.close
    }
  }
```

```
  def printNullModelResults(cScoreObs: Double,
                            cScoreExp: List[Double],
                            mNNObs: Double,
                            mSDNNObs: Double,
                            mAITS_NNExp: List[Double],
                            mAITS_SDNNExp: List[Double],
                            mAWTS_NNExp: List[Double],
                            mAWTS_SDNNExp: List[Double],
                            maxC: Double,
                            species: Int,
                            plots: Int,
                            X: Int,
                            Y: Int,
                            co_iter: Int,
                            x: Int,
                            outfile: String) {

    val resultString =
      (x + 1).toString + "\t" +
        formatter.format(cScoreObs) + "\t" +
        formatter.format(stat.mean(cScoreExp)) + "\t" +
        formatter.format(stat.stdDev(cScoreExp)) + "\t" +
        formatter.format(cScoreExp.min) + "\t" +
        formatter.format(cScoreExp.max) + "\t" +
        pValue.format(cScoreExp.count(_ < cScoreObs).toDouble / cScoreExp.length.toDouble) + "\t" +
        pValue.format(cScoreExp.count(_ == cScoreObs).toDouble / cScoreExp.length.toDouble) + "\t" +
        pValue.format(cScoreExp.count(_ > cScoreObs).toDouble / cScoreExp.length.toDouble) + "\t" +
        pValue.format((cScoreObs - stat.mean(cScoreExp)) / stat.stdDev(cScoreExp)) + "\t" +
        formatter.format(mNNObs) + "\t" +
        formatter.format(stat.mean(mAITS_NNExp)) + "\t" +
        formatter.format(stat.stdDev(mAITS_NNExp)) + "\t" +
        formatter.format(mAITS_NNExp.min) + "\t" +
        formatter.format(mAITS_NNExp.max) + "\t" +
        pValue.format(mAITS_NNExp.count(_ < mNNObs).toDouble / mAITS_NNExp.length.toDouble) + "\t" +
        pValue.format(mAITS_NNExp.count(_ == mNNObs).toDouble / mAITS_NNExp.length.toDouble) + "\t" +
        pValue.format(mAITS_NNExp.count(_ > mNNObs).toDouble / mAITS_NNExp.length.toDouble) + "\t" +
        pValue.format((mNNObs - stat.mean(mAITS_NNExp)) / stat.stdDev(mAITS_NNExp)) + "\t" +
        formatter.format(mAWTS_NNExp.min) + "\t" +
        formatter.format(mAWTS_NNExp.max) + "\t" +
        pValue.format(mAWTS_NNExp.count(_ < mNNObs).toDouble / mAWTS_NNExp.length.toDouble) + "\t" +
        pValue.format(mAWTS_NNExp.count(_ == mNNObs).toDouble / mAWTS_NNExp.length.toDouble) + "\t" +
        pValue.format(mAWTS_NNExp.count(_ > mNNObs).toDouble / mAWTS_NNExp.length.toDouble) + "\t" +
        pValue.format((mNNObs - stat.mean(mAWTS_NNExp)) / stat.stdDev(mAWTS_NNExp)) + "\t" +
        formatter.format(mSDNNObs) + "\t" +
        formatter.format(stat.mean(mAITS_SDNNExp)) + "\t" +
        formatter.format(stat.stdDev(mAITS_SDNNExp)) + "\t" +
        formatter.format(mAITS_SDNNExp.min) + "\t" +
        formatter.format(mAITS_SDNNExp.max) + "\t" +
        pValue.format(mAITS_SDNNExp.count(_ < mSDNNObs).toDouble / mAITS_SDNNExp.length.toDouble) + "\t" +
        pValue.format(mAITS_SDNNExp.count(_ == mSDNNObs).toDouble / mAITS_SDNNExp.length.toDouble) + "\t"
+
        pValue.format(mAITS_SDNNExp.count(_ > mSDNNObs).toDouble / mAITS_SDNNExp.length.toDouble) + "\t" +
```

```
      pValue.format((mSDNNObs - stat.mean(mAITS_SDNNExp)) / stat.stdDev(mAITS_SDNNExp)) + "\t" +
      formatter.format(stat.mean(mAWTS_SDNNExp)) + "\t" +
      formatter.format(stat.stdDev(mAWTS_SDNNExp)) + "\t" +
      formatter.format(mAWTS_SDNNExp.min) + "\t" +
      formatter.format(mAWTS_SDNNExp.max) + "\t" +
      pValue.format(mAWTS_SDNNExp.count(_ < mSDNNObs).toDouble / mAWTS_SDNNExp.length.toDouble) +
"\t" +
      pValue.format(mAWTS_SDNNExp.count(_ == mSDNNObs).toDouble / mAWTS_SDNNExp.length.toDouble) +
"\t" +
      pValue.format(mAWTS_SDNNExp.count(_ > mSDNNObs).toDouble / mAWTS_SDNNExp.length.toDouble) +
"\t" +
      pValue.format((mSDNNObs - stat.mean(mAWTS_SDNNExp)) / stat.stdDev(mAWTS_SDNNExp)) + "\t" +
      formatter.format(maxC) + "\t" +
      pValue.format(cScoreObs / maxC) + "\t" +
      species + "\t" + plots + "\t" + species * plots

    if (currentFile != outfile) {
     val runParm =
      "Run parameters:\n" +
       "  Species - " + species + "\n" +
       "  Plots   - " + plots + "\n" +
       "  X       - " + X + "\n" +
       "  Y       - " + Y + "\n" +
       "  co_iter - " + co_iter + "\n"
     val strHeader =
      "Run#\t" +
       "ObsC\t" +
       "MeanC\t" +
       "Std_C\t" +
       "Min_C\t" +
       "Max_C\t" +
       "lt_C\t" +
       "eq_C\t" +
       "gt_C\t" +
       "SES_C\t" +
       "nnObs\t" +
       "AITS_mNN\t" +
       "AITS_Std_NN\t" +
       "AITS_Min_NN\t" +
       "AITS_Max_NN\t" +
       "AITS_lt_NN\t" +
       "AITS_eq_NN\t" +
       "AITS_gt_NN\t" +
       "AITS_SES_NN\t" +
       "AWTS_Min_NN\t" +
       "AWTS_Max_NN\t" +
       "AWTS_lt_NN\t" +
       "AWTS_eq_NN\t" +
       "AWTS_gt_NN\t" +
       "AWTS_SES_NN\t" +
       "sdnnObs\t" +
       "AITS_mSDNN\t" +
       "AITS_Std_SDNN\t" +
```

```
        "AITS_Min_SDNN\t" +
        "AITS_Max_SDNN\t" +
        "AITS_lt_SDNN\t" +
        "AITS_eq_SDNN\t" +
        "AITS_gt_SDNN\t" +
        "AITS_SES_SDNN\t" +
        "AWTS_mSDNN\t" +
        "AWTS_Std_SDNN\t" +
        "AWTS_Min_SDNN\t" +
        "AWTS_Max_SDNN\t" +
        "AWTS_lt_SDNN\t" +
        "AWTS_eq_SDNN\t" +
        "AWTS_gt_SDNN\t" +
        "AWTS_SES_SDNN\t" +
        "maxC\t" +
        "PercentC\t" +
        "Species\t" +
        "Plots\t" +
        "Dim"
      if (!silent) println(runParm)
      //     writeToFile(outfile, runParm)
      if (!silent) println(strHeader)
      writeToFile(outfile, strHeader)
      currentFile = outfile
    }
    if (!silent) println(resultString)
    writeToFile(outfile, resultString)
  }

}
```

B.1.16 RowShuffle.scala

```
package ca.mikelavender.chapterone


class RowShuffle {

  def shuffle(matrix: Matrix, iterations: Int = 60000) = {
    var counter = 0
    val rowCount = matrix.size
    val colCount = matrix(0).size
    def loop: Matrix = counter match {
      case `iterations` => null
      case _ =>
        counter += 1
        val cols = getNextIndex(rnd.nextInt(colCount), colCount)
        swap(matrix(rnd.nextInt(rowCount)), cols._1, cols._2)
        loop
    }
    loop
```

```
    matrix
  }

  def swap(matrix: Array[Int], a: Int, b: Int) {
    val t = matrix(a)
    matrix(a) = matrix(b)
    matrix(b) = t
  }

  def getNextIndex(i: Int, constraint: Int): Tuple2[Int, Int] = {
    val newIndex = rnd.nextInt(constraint)
    newIndex match {
      case `i` => getNextIndex(i, constraint)
      case _ => (i, newIndex)
    }
  }
}
```

B.1.17 Statistics.scala

```
package ca.mikelavender.chapterone

import math._


class Statistics {

  def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

  def stdDev(numbers: List[Double]): Double = {

    var sum: Double = 0.0
    if (numbers.length >= 2) {
      val avg = mean(numbers)
      val factor: Double = 1.0 / (numbers.length.toDouble - 1)
      for (x: Double <- numbers) {
        sum = sum + ((x - avg) * (x - avg))
      }
      sum = sum * factor
    }
    sqrt(sum)
  }

}
```

B.1.18 Utilities.scala

```
package ca.mikelavender.chapterone
```

```
import java.util.concurrent.TimeUnit


class Utilities {

  def timer(f: => Unit) = {
    val s = System.currentTimeMillis
    f
    val stop = System.currentTimeMillis - s

    "%d min, %d sec: %d".format(
      TimeUnit.MILLISECONDS.toMinutes(stop),
      TimeUnit.MILLISECONDS.toSeconds(stop) -
        TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(stop)), stop
    )
  }

  def optTimer(f: => Unit) = {
    val s = System.currentTimeMillis
    f
    System.currentTimeMillis - s
  }

  def printMatrix(matrix: Array[Array[Int]]) {
    for (i <- 0 to matrix.size - 1) {
      for (j <- 0 to matrix(i).size - 1) {
        print(matrix(i)(j) + " ")
      }
      print("\n")
    }
    println("--------------------------------\n")
  }

  def printMatrix(matrix: Array[Array[Double]]) {
    for (i <- 0 to matrix.size - 1) {
      for (j <- 0 to matrix(i).size - 1) {
        print(matrix(i)(j) + " ")
      }
      print("\n")
    }
    println("--------------------------------\n")
  }

}
```

B.1.19 WeightedRandomSelection.scala

```
package ca.mikelavender.chapterone

import util.Random
```

```
object WeightedRandomSelection {

  /**
   * Get the number of times an event with probability p occurs in N samples.
   * if R is res, then P(R=n) = p^n q^(N-n) N! / n! / (N-n)!
   * where q = 1-p
   * This has the property that P(R=0) = q^N, and
   * P(R=n+1) = p/q (N-n)/(n+1) P(R=n)
   * Also note that P(R=n+1|R>n) = P(R=n+1)/P(R>n)
   * Uses these facts to work out the probability that the result is zero. If
   * not, then the prob that given that, the result is 1, etc.
   */
  def numEntries(p: Double, N: Int, r: Random): Int = if (p > 0.5) N -
    numEntries(1.0 - p, N, r)
  else if (p < 0.0) 0
  else {
    val q = 1.0 - p
    if (N * p > 100.0) {
      // numeric underflow can be a problem with exact computation, but gaussian approximation is good
      val continuousApprox = r.nextGaussian() * math.sqrt(N * p * q) + N * p
      continuousApprox.round.toInt.max(0).min(N)
    } else {
      // exact approx will not underflow
      var n = 0
      var prstop = math.pow(q, N)
      var cumulative = 0.0
      while (n < N && (r.nextDouble() * (1 - cumulative)) >= prstop) {
        cumulative += prstop
        prstop *= p * (N - n) / (q * (n + 1))
        n += 1
      }
      n
    }
  }


  case class WeightedItem[T](item: T, weight: Double)

  /**
   * Compute a weighted selection from the given items.
   * cumulativeSum must be the same length as items (or longer), with the ith element containing the sum of all
   * weights from the item i to the end of the list. This is done in a saved way rather than adding up and then
   * subtracting in order to prevent rounding errors from causing a variety of subtle problems.
   */
  private def weightedSelectionWithCumSum[T](items: Seq[WeightedItem[T]], cumulativeSum: List[Double],
numSelections: Int, r: Random): Seq[T] = {
    if (numSelections == 0) Nil
    else {
      val head = items.head
      val nhead = numEntries(head.weight / cumulativeSum.head, numSelections, r)
      List.fill(nhead)(head.item) ++ weightedSelectionWithCumSum(items.tail, cumulativeSum.tail, numSelections -
nhead, r)
    }
```

```
  }


  def weightedSelection[T](items: Seq[WeightedItem[T]], numSelections: Int, r: Random): Seq[T] = {
    val cumsum = items.foldRight(List(0.0)) {
      (wi, l) => (wi.weight + l.head) :: l
    }
    weightedSelectionWithCumSum(items, cumsum, numSelections, r)
  }


  def testRandomness[T](items: Seq[WeightedItem[T]], numSelections: Int, r: Random) {
    val runs = 5000
    val indexOfItem = Map.empty ++ items.zipWithIndex.map {
      case (item, ind) => item.item -> ind
    }
    val numItems = items.length
    val bucketSums = new Array[Double](numItems)
    val bucketSumSqs = new Array[Double](numItems)
    for (run <- 0 until runs) {
      // compute chi-squared for a run
      val runresult = weightedSelection(items, numSelections, r)
      val buckets = new Array[Double](numItems)
      for (r <- runresult) buckets(indexOfItem(r)) += 1
      for (i <- 0 until numItems) {
        val count = buckets(i)
        bucketSums(i) += count
        bucketSumSqs(i) += count * count
      }
    }
    val sumWeights = items.foldLeft(0.0)(_ + _.weight)
    for ((item, ind) <- items.zipWithIndex) {
      val p = item.weight / sumWeights
      val mean = bucketSums(ind) / runs
      val variance = bucketSumSqs(ind) / runs - mean * mean
      val expectedMean = numSelections * p
      val expectedVariance = numSelections * p * (1 - p)
      val expectedErrorInMean = math.sqrt(expectedVariance / runs)
      val text = "Item %10s Mean %.3f Expected %.3f±%.3f Variance %.3f expected %.3f".format(item.item, mean,
expectedMean, expectedErrorInMean, variance, expectedVariance)
      println(text)
    }
  }

  def main(args: Array[String]): Unit = {
    val items = Seq(WeightedItem("Red", 1d / 6), WeightedItem("Blue", 2d / 6), WeightedItem("Green", 3d / 6))
    println(weightedSelection(items, 6, new Random()))
    testRandomness(items, 6, new Random())
  }

}
```

B.2 Scala scripts

B.2.1 ConvertToSingleFile.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}

/**
 * File name: ConvertToSingleFile.scala
 *
 * This script converts all of the type I data files into one single file.
 * The output file (TypeIAll.tsv) is consumed by the R script 'C Score SES x Trait SES'
 * to generate the cluster plot of SES values.
 *
 */


val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type I Error Rate Data"

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

val header =
"Run#\tObsC\tMeanC\tStd_C\tMin_C\tMax_C\tlt_C\teq_C\tgt_C\tSES_C\tnnObs\tAITS_mNN\tAITS_Std_NN\t
AITS_Min_NN\tAITS_Max_NN\tAITS_lt_NN\tAITS_eq_NN\tAITS_gt_NN\tAITS_SES_NN\tAWTS_Min_NN\tAWTS
_Max_NN\tAWTS_lt_NN\tAWTS_eq_NN\tAWTS_gt_NN\tAWTS_SES_NN\tsdnnObs\tAITS_mSDNN\tAITS_Std_S
DNN\tAITS_Min_SDNN\tAITS_Max_SDNN\tAITS_lt_SDNN\tAITS_eq_SDNN\tAITS_gt_SDNN\tAITS_SES_SDNN\t
AWTS_mSDNN\tAWTS_Std_SDNN\tAWTS_Min_SDNN\tAWTS_Max_SDNN\tAWTS_lt_SDNN\tAWTS_eq_SDNN\
tAWTS_gt_SDNN\tAWTS_SES_SDNN\tmaxC\tPercentC\tSpecies\tPlots\tDim"

val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/TypeIAll.tsv"))

f.write("ObsC\tSES_C\tAITS_SES_NN\tAWTS_SES_NN\tAITS_SES_SDNN\tAWTS_SES_SDNN\tSpecies\tPlots\tDi
m" + "\n")

val hMap = buildMap(header)

for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines().drop(1)

  for (line <- lines) {
    val currentLine = line split ("\t")
    val obsC = currentLine(hMap("ObsC"))
    val c_SES = currentLine(hMap("SES_C"))
    val mNTDaitsSES = currentLine(hMap("AITS_SES_NN"))
    val mNTDawtsSES = currentLine(hMap("AWTS_SES_NN"))
```

```
    val sDNNaitsSES = currentLine(hMap("AITS_SES_SDNN"))
    val sDNNawtsSES = currentLine(hMap("AWTS_SES_SDNN"))
    val dim = currentLine(hMap("Dim"))
    val species = currentLine(hMap("Species"))
    val plots = currentLine(hMap("Plots"))
    f.write(obsC + "\t" +
      c_SES + "\t" +
      mNTDaitsSES + "\t" +
      mNTDawtsSES + "\t" +
      sDNNaitsSES + "\t" +
      sDNNawtsSES + "\t" +
      species + "\t" +
      plots + "\t" +
      dim + "\n"
    )
  }

  f.flush()

}

f.close()

def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.2 LogisticRegression.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}

/**
 * File name: LogisticRegression.scala
 * Consumed By: logRegression x Trait SES.R
 *
 * This script parses all of the type I error rate data and produces a single
 * monolithic file for R to consume. The purpose of this data is to test
 * the relationship between CO SES values and LS SES values.
 */


val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type I Error Rate Data"

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

def isSignificant(ses: String) = {
  val v = ses match {
    case """  """ => Double.PositiveInfinity
    case """?""" => Double.PositiveInfinity
```

```
    case """-?""" => Double.NegativeInfinity
    case _ => ses.toDouble
  }

  if (v == Double.PositiveInfinity) "Inf"
  else if (v == Double.NegativeInfinity) "-Inf"
  else v
 }

 def dLine(currentLine: Array[String]): Array[String] = {
  currentLine.map {
    case """?""" => "Inf"
    case """?""" => "Inf"
    case """-?""" => "-Inf"
    case """-?""" => "-Inf"
    case s => s
  }
 }

 val header =
"Run#\tObsC\tMeanC\tStd_C\tMin_C\tMax_C\tlt_C\teq_C\tgt_C\tSES_C\tnnObs\tAITS_mNN\tAITS_Std_NN\t
AITS_Min_NN\tAITS_Max_NN\tAITS_lt_NN\tAITS_eq_NN\tAITS_gt_NN\tAITS_SES_NN\tAWTS_Min_NN\tAWTS
_Max_NN\tAWTS_lt_NN\tAWTS_eq_NN\tAWTS_gt_NN\tAWTS_SES_NN\tsdnnObs\tAITS_mSDNN\tAITS_Std_S
DNN\tAITS_Min_SDNN\tAITS_Max_SDNN\tAITS_lt_SDNN\tAITS_eq_SDNN\tAITS_gt_SDNN\tAITS_SES_SDNN\t
AWTS_mSDNN\tAWTS_Std_SDNN\tAWTS_Min_SDNN\tAWTS_Max_SDNN\tAWTS_lt_SDNN\tAWTS_eq_SDNN\
tAWTS_gt_SDNN\tAWTS_SES_SDNN\tmaxC\tPercentC\tSpecies\tPlots\tDim"

 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/TypeIAllOrdered.tsv"))

f.write("ObsC\tSES_C\tAITS_SES_NN\tAWTS_SES_NN\tAITS_SES_SDNN\tAWTS_SES_SDNN\tSpecies\tPlots\tDi
m" + "\n")

 val hMap = buildMap(header)

 for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines().drop(1)

  for (line <- lines) {
    val currentLine = dLine(line.split("\t"))
    val obsC = currentLine(hMap("ObsC"))
    val c_SES = currentLine(hMap("SES_C"))
    val mNTDaitsSES = currentLine(hMap("AITS_SES_NN"))
    val mNTDawtsSES = currentLine(hMap("AWTS_SES_NN"))
    val sDNNaitsSES = currentLine(hMap("AITS_SES_SDNN"))
    val sDNNawtsSES = currentLine(hMap("AWTS_SES_SDNN"))
    val dim = currentLine(hMap("Dim"))
    val species = currentLine(hMap("Species"))
    val plots = currentLine(hMap("Plots"))
    f.write(obsC + "\t" +
      c_SES + "\t" +
```

```
      mNTDaitsSES + "\t" +
      mNTDawtsSES + "\t" +
      sDNNaitsSES + "\t" +
      sDNNawtsSES + "\t" +
      species + "\t" +
      plots + "\t" +
      dim + "\n"
    )
  }

  f.flush()

}

f.close()

def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.3 OccurrenceVsAbundance.scala

```
import java.io.{FileWriter, BufferedWriter, File}
import java.text.DecimalFormat
import scala.io.Source

val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/ISData/nn"
var ps = 0d
val abundanceMap = new scala.collection.mutable.HashMap[Int, Double]
val formatter = new DecimalFormat("#0.000000")
val percent = new DecimalFormat("#0.00")

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  // f.getName.endsWith("sp20p50perm100000N15Type-nn_1455_SubsetsFinal.v1.log"))
  f.getName.endsWith("_SubsetsFinal.v1.log"))
  .toIterator


var map = Map[(Int, Int, Int, String, Int), (Int, Int)]()

val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
3/Summarized Data/AbundanceVsOccurrence.tsv"))
val header = "num.sp" + "\t" +
  "num.pl" + "\t" +
  "N" + "\t" +
  "subSize" + "\t" +
  "spId" + "\t" +
  "spAbund" + "\t" +
  "spFreq" + "\t" +
  "normalized" + "\t" +
  "fileID"
```

```
f.write(header + "\n")
println("Processing")

val fileCount = getFiles(path).size.toDouble
var counter = 0

for (file <- getFiles(path)) {
  counter += 1
  //  println(file.getName)

  // parse file name to get sp, p, N and type
  val regex =
"""sp(20|40|60|80|100)p(20|25|50|75|40|60|80|100|150|200)perm100000N(0|5|10|15|20)Type-
(nn|sdnn|co)_[0-9]{1,5}_SubsetsFinal\.v1\.log""".r
  val regex(sp, pl, n, typ) = file.getName
  //  println("sp: " + sp + "\tpl: " + pl + "\tn: " + n + "\ttyp: " + typ)
  ps = pl.toDouble

  val parsedFile = readFile(file)


  val groups = parsedFile.groupBy(a => a.length)


  for (groupTuple <- groups.filter(_._2.length > 500)) {
    val group = groupTuple._2
    val subsetSize = groupTuple._1
    val prob = 1d / sp.toDouble * subsetSize.toDouble
    val frequencyMap = buildItemFrequenciesMap(group.map(_.toList)).withDefaultValue(0)

    val A = frequencyMap.minBy(_._2)._2.toDouble
    val B = frequencyMap.maxBy(_._2)._2.toDouble

    val a = -1d
    val b = 1d


    for (i <- 0 to sp.toInt - 1) {
      val string = sp + "\t" +
        pl + "\t" +
        n + "\t" +
        subsetSize + "\t" +
        i + "\t" +
        formatter.format(abundanceMap(i)) + "\t" +
        formatter.format(frequencyMap(i).toDouble / group.length.toDouble) + "\t" +
        formatter.format(frequencyMap(i).toDouble / group.length.toDouble - prob)  + "\t" +
        file.getName

      f.write(string + "\n")
    }
  }
  print("\r" + percent.format(counter.toDouble / fileCount * 100) + "% complete")
}
```

```
println
f.flush()
f.close()

def readFile(infile: File) = {
  var hMap = Map[String, Int]()
  var parsedFile = List[Array[Int]]()
  val lines = Source.fromFile(infile).getLines()
  var pastMatrix = false

  for (line <- lines) {

    if (line.contains(":\t")) {
      val currentLine = line.split(":\t")
      abundanceMap.put(currentLine(0).toInt, currentLine(1).split(",").count(_ == "1").toDouble / ps)

      //     println("sp " + currentLine(0) + ":\t " + currentLine(1).split(",").count(_ == "1").toDouble / ps)
    }

    if (line.contains("##")) pastMatrix = true

    if (line.contains("Sp_Group")) {
      hMap = buildMap(line)
    } else if (pastMatrix & line.split("\t").length == 8) {
      val currentLine = line split "\t"
      if (currentLine(hMap("AITS_NN")).contains("gt")) parsedFile = extractToList(currentLine(hMap("Sp_Group")))
:: parsedFile
    }
  }

  def buildMap(line: String) = line.split("\t").zipWithIndex.toMap

  def extractToList(s: String): Array[Int] = {
    if (s == "List()") Array[Int]()
    else {
      s.replaceAll( """(List\(|\)| )""", "")
        .split(",")
        .map(_.toInt)
    }
  }
  parsedFile
}

def buildItemFrequenciesMap(dataSource: List[List[Int]]) = {
  val freqMap = new scala.collection.mutable.HashMap[Int, Int]
  dataSource.foreach((trans) => {
    trans.foreach((x) => {
      if (!freqMap.contains(x)) {
        freqMap.put(x, 1)
      } else {
        freqMap(x) += 1
      }
    })
```

```
  })

  freqMap
 }

 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.4 ParseFrequencies.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}
import java.text.DecimalFormat
import math._



var map: Map[Int, List[Int]] = Map()
val pValue = new DecimalFormat("#0.0")
val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/frequencies.csv"
val file = new File(path)

val lines = Source.fromFile(file).getLines().drop(1)

for (line <- lines) {
  val currentLine = line.split(", ")
  if (map.contains(currentLine(4).toInt)) {
    map += currentLine(4).toInt -> {
      currentLine(1).toInt :: map(currentLine(4).toInt)
    }
  } else {
    map += currentLine(4).toInt -> List(currentLine(1).toInt)
  }
}

val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/MatrixRepeats.tsv"))
f.write("Dim\tmean\tnegSTDev\tposSTDev" + "\n")
println("Dim\tmean\tnegSTDev\tposSTDev")

for (key <- map.keySet.toList.sorted) {
  println(key + "\t" +
    pValue.format(mean(map(key))) + "\t" +
    pValue.format(mean(map(key)) + stdDev(map(key))) + "\t" +
    pValue.format(mean(map(key)) - stdDev(map(key))))

  f.write(key + "\t" +
    pValue.format(mean(map(key))) + "\t" +
    pValue.format(mean(map(key)) + stdDev(map(key))) + "\t" +
    pValue.format(mean(map(key)) - stdDev(map(key))) + "\n")

  f.flush
```

```
}

f.flush()
f.close()

def mean(numbers: List[Int]) = numbers.reduceLeft(_ + _).toDouble / numbers.length

def stdDev(numbers: List[Int]) = {

  var sum: Double = 0.0
  if (numbers.length >= 2) {
    val avg = mean(numbers)
    val factor: Double = 1.0 / (numbers.length.toDouble - 1)
    for (x <- numbers) {
      sum = sum + ((x - avg) * (x - avg))
    }
    sum = sum * factor
  }
  sqrt(sum)
}


def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.5 AllTypeIColour.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}
import java.text.DecimalFormat

val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type I Error Rate Data"

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

def checkSig(lt: Double, gt: Double) = if (lt >= 0.975 || gt >= 0.975) 1 else 0
def checkEqual(lt: Double, eq: Double, gt: Double) = if (lt + eq >= 0.975 || gt + eq >= 0.975) 1 else 0
def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

var dimMap, plMap, spMap = Map[(Int, Int), (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int)]()

for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(7)

  var hFlag = true
  var hMap = Map[String, Int]()
```

```
  for (line <- lines) {
   if (hFlag) {
    hMap = buildMap(line)
    hFlag = false
   } else {
    val currentLine = line.split("\t")
    val plots = currentLine(hMap("Plots")).toInt
    val species = currentLine(hMap("Species")).toInt
    val dim = plots * species
    val dLine = currentLine.map(s => s match {
     case """"""" => Double.PositiveInfinity
     case """?""" => Double.PositiveInfinity
     case """∞""" => Double.PositiveInfinity
     case """-""" => Double.NegativeInfinity
     case """-?""" => Double.NegativeInfinity
     case """-∞""" => Double.NegativeInfinity
     case _ => s.toDouble
    })

    if (!dimMap.contains((species, plots))) {
     dimMap += (species, plots) -> {
      (1,
       checkEqual(dLine(hMap("lt_C")), dLine(hMap("eq_C")), dLine(hMap("gt_C"))),
       checkSig(dLine(hMap("lt_C")), dLine(hMap("gt_C"))),
       checkSES(dLine(hMap("SES_C"))),
       checkEqual(dLine(hMap("AITS_lt_NN")), dLine(hMap("AITS_eq_NN")), dLine(hMap("AITS_gt_NN"))),
       checkSig(dLine(hMap("AITS_lt_NN")), dLine(hMap("AITS_gt_NN"))),
       checkSES(dLine(hMap("AITS_SES_NN"))),
       checkEqual(dLine(hMap("AWTS_lt_NN")), dLine(hMap("AWTS_eq_NN")), dLine(hMap("AWTS_gt_NN"))),
       checkSig(dLine(hMap("AWTS_lt_NN")), dLine(hMap("AWTS_gt_NN"))),
       checkSES(dLine(hMap("AWTS_SES_NN"))),
       checkEqual(dLine(hMap("AITS_lt_SDNN")), dLine(hMap("AITS_eq_SDNN")),
dLine(hMap("AITS_gt_SDNN"))),
       checkSig(dLine(hMap("AITS_lt_SDNN")), dLine(hMap("AITS_gt_SDNN"))),
       checkSES(dLine(hMap("AITS_SES_SDNN"))),
       checkEqual(dLine(hMap("AWTS_lt_SDNN")), dLine(hMap("AWTS_eq_SDNN")),
dLine(hMap("AWTS_gt_SDNN"))),
       checkSig(dLine(hMap("AWTS_lt_SDNN")), dLine(hMap("AWTS_gt_SDNN"))),
       checkSES(dLine(hMap("AWTS_SES_SDNN")))
       )
     }
    } else {
     dimMap += (species, plots) -> {
      (dimMap((species, plots))._1 + 1,
       dimMap((species, plots))._2 + checkEqual(dLine(hMap("lt_C")), dLine(hMap("eq_C")),
dLine(hMap("gt_C"))),
       dimMap((species, plots))._3 + checkSig(dLine(hMap("lt_C")), dLine(hMap("gt_C"))),
       dimMap((species, plots))._4 + checkSES(dLine(hMap("SES_C"))),
       dimMap((species, plots))._5 + checkEqual(dLine(hMap("AITS_lt_NN")), dLine(hMap("AITS_eq_NN")),
dLine(hMap("AITS_gt_NN"))),
       dimMap((species, plots))._6 + checkSig(dLine(hMap("AITS_lt_NN")), dLine(hMap("AITS_gt_NN"))),
       dimMap((species, plots))._7 + checkSES(dLine(hMap("AITS_SES_NN"))),
```

```
        dimMap((species, plots))._8 + checkEqual(dLine(hMap("AWTS_lt_NN")), dLine(hMap("AWTS_eq_NN")),
dLine(hMap("AWTS_gt_NN"))),
        dimMap((species, plots))._9 + checkSig(dLine(hMap("AWTS_lt_NN")), dLine(hMap("AWTS_gt_NN"))),
        dimMap((species, plots))._10 + checkSES(dLine(hMap("AWTS_SES_NN"))),
        dimMap((species, plots))._11 + checkEqual(dLine(hMap("AITS_lt_SDNN")),
dLine(hMap("AITS_eq_SDNN")), dLine(hMap("AITS_gt_SDNN"))),
        dimMap((species, plots))._12 + checkSig(dLine(hMap("AITS_lt_SDNN")), dLine(hMap("AITS_gt_SDNN"))),
        dimMap((species, plots))._13 + checkSES(dLine(hMap("AITS_SES_SDNN"))),
        dimMap((species, plots))._14 + checkEqual(dLine(hMap("AWTS_lt_SDNN")),
dLine(hMap("AWTS_eq_SDNN")), dLine(hMap("AWTS_gt_SDNN"))),
        dimMap((species, plots))._15 + checkSig(dLine(hMap("AWTS_lt_SDNN")),
dLine(hMap("AWTS_gt_SDNN"))),
        dimMap((species, plots))._16 + checkSES(dLine(hMap("AWTS_SES_SDNN")))
        )
      }
     }
    }
   }
  }

 val pValue = new DecimalFormat("#0.000000")

 val f1 = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/CoOccurTypeIColour.tsv"))
 f1.write("Species\tPlots\tCount\ttype\tval\n")

 println("Species\tPlots\tCount\ttype\tval")
 for (key <- dimMap.keySet.toList.sorted) {
  println(
   key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "C_eq" + "\t" + pValue.format(dimMap(key)._2 /
dimMap(key)._1.toDouble) + "\n" +
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "C_ne" + "\t" + pValue.format(dimMap(key)._3 /
dimMap(key)._1.toDouble) + "\n" +
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "SES" + "\t" + pValue.format(dimMap(key)._4 /
dimMap(key)._1.toDouble))

  f1.write(
   key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "C_eq" + "\t" + pValue.format(dimMap(key)._2 /
dimMap(key)._1.toDouble) + "\n" +
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "C_ne" + "\t" + pValue.format(dimMap(key)._3 /
dimMap(key)._1.toDouble) + "\n" +
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "SES" + "\t" + pValue.format(dimMap(key)._4 /
dimMap(key)._1.toDouble) + "\n")
  f1.flush()
 }

 f1.flush()
 f1.close()


 val f2 = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/TraitTypeIColour.tsv"))
 f2.write("Species\tPlots\tCount\ttype\tval\n")
```

```
println("Species\tPlots\tCount\ttype\tval")
for (key <- dimMap.keySet.toList.sorted) {
  println(
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSntd_eq" + "\t" + pValue.format(dimMap(key)._5 /
dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSntd_ne" + "\t" + pValue.format(dimMap(key)._6
/ dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSntd_SES" + "\t" + pValue.format(dimMap(key)._7
/ dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSntd_eq" + "\t" +
pValue.format(dimMap(key)._8 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSntd_ne" + "\t" +
pValue.format(dimMap(key)._9 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSntd_SES" + "\t" +
pValue.format(dimMap(key)._10 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSsdnn_eq" + "\t" +
pValue.format(dimMap(key)._11 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSsdnn_ne" + "\t" +
pValue.format(dimMap(key)._12 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSsdnn_SES" + "\t" +
pValue.format(dimMap(key)._13 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSsdnn_eq" + "\t" +
pValue.format(dimMap(key)._14 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSsdnn_ne" + "\t" +
pValue.format(dimMap(key)._15 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSsdnn_SES" + "\t" +
pValue.format(dimMap(key)._16 / dimMap(key)._1.toDouble))

  f2.write(
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSntd_eq" + "\t" + pValue.format(dimMap(key)._5 /
dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSntd_ne" + "\t" + pValue.format(dimMap(key)._6
/ dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSntd_SES" + "\t" + pValue.format(dimMap(key)._7
/ dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSntd_eq" + "\t" +
pValue.format(dimMap(key)._8 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSntd_ne" + "\t" +
pValue.format(dimMap(key)._9 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSntd_SES" + "\t" +
pValue.format(dimMap(key)._10 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSsdnn_eq" + "\t" +
pValue.format(dimMap(key)._11 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSsdnn_ne" + "\t" +
pValue.format(dimMap(key)._12 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AITSsdnn_SES" + "\t" +
pValue.format(dimMap(key)._13 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSsdnn_eq" + "\t" +
pValue.format(dimMap(key)._14 / dimMap(key)._1.toDouble) + "\n" +
      key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSsdnn_ne" + "\t" +
pValue.format(dimMap(key)._15 / dimMap(key)._1.toDouble) + "\n" +
```

```
    key._1 + "\t" + key._2 + "\t" + dimMap(key)._1 + "\t" + "AWTSsdnn_SES" + "\t" +
pValue.format(dimMap(key)._16 / dimMap(key)._1.toDouble) + "\n")
  f2.flush()
 }

 f2.flush()
 f2.close()


 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.6 CScore_TypeIIColour.scala

```
 import io.Source
 import java.io.{FileWriter, BufferedWriter, File}

 /*
 *
 * This script parses the Type II sensitivity data for the co-occurrence results and summerises it for R.
 * The final product is COTypeIISensitivitySummary.csv which is used to produce
 * the Type II Plot x Dim Panels plot.
 *
 * */

 val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/CO"

 def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

 def oneTailedExclusive(lt: Double) = if (lt >= 0.95) 1 else 0
 def oneTailedInclusive(lt: Double, eq: Double) = if (lt + eq >= 0.975) 1 else 0
 def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

 var dimMap = Map[(Int, Int), (Int, Int, Int, Int)]()
 val propRange = List(100)

 for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines; if (line split ("\t")).length == 13) {
   if (hFlag) {
    hMap = buildMap(line)
    hFlag = false
   } else {
    val currentLine = line split ("\t")
```

```
    val n = currentLine(hMap("N")).toInt
    val plots = currentLine(hMap("Plots")).toInt
    val species = currentLine(hMap("Species")).toInt
    val dim = currentLine(hMap("Dim")).toInt
    val proportion = ((n.toDouble / species.toDouble) * 100).toInt
    val dLine = currentLine.map(s => s match {
     case """�""" => Double.PositiveInfinity
     case """?""" => Double.PositiveInfinity
     case """∞""" => Double.PositiveInfinity
     case """-�""" => Double.NegativeInfinity
     case """-?""" => Double.NegativeInfinity
     case """-∞""" => Double.NegativeInfinity
     case _ => s.toDouble
    })

    "id\tcBefore\tobsC\texpC\texpCStd\tlt\teq\tgt\tses\tSpecies\tPlots\tDim\tN"


    if (proportion == 100) {

     if (!dimMap.contains(species, plots)) {
      dimMap += (species, plots) -> {
       (1,
        oneTailedExclusive(dLine(hMap("lt"))),
        oneTailedInclusive(dLine(hMap("lt")), dLine(hMap("eq"))),
        checkSES(dLine(hMap("ses")))
        )
      }
     } else {
      dimMap += (species, plots) -> {
       (dimMap((species, plots))._1 + 1,
        dimMap((species, plots))._2 + oneTailedExclusive(dLine(hMap("lt"))),
        dimMap((species, plots))._3 + oneTailedInclusive(dLine(hMap("lt")), dLine(hMap("eq"))),
        dimMap((species, plots))._4 + checkSES(dLine(hMap("ses")))
        )
      }
     }
    }
   }
  }
 }

 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/COTypeIIColour.csv"))
 f.write("Species,Plots,Count,type,val" + "\n")

 println("\nSpecies\tPlots\tCount\ttype\tval")
 for (key <- dimMap.keySet.toList.sorted) {
  println(
   key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teq" + "\t" +
dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tne" + "\t" +
dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
```

```
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tses" + "\t" +
dimMap(key)._4.toDouble / dimMap(key)._1
  )

  f.write(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", ",") + "," + dimMap(key)._1 + ",eq" + "," +
dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
    key.toString().replaceAll( """[\(\)]""", "").replace(",", ",") + "," + dimMap(key)._1 + ",ne" + "," +
dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
    key.toString().replaceAll( """[\(\)]""", "").replace(",", ",") + "," + dimMap(key)._1 + ",ses" + "," +
dimMap(key)._4.toDouble / dimMap(key)._1 + "\n"
  )

  f.flush()
 }

 f.flush()
 f.close()

 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.7 CScore_TypeIIColourExtended.scala

```
 import io.Source
 import java.io.{FileWriter, BufferedWriter, File}

 /*
 *
 * This script parses the Type II sensitivity data for the co-occurrence results and summerises it for R.
 * The final product is COTypeIISensitivitySummary.csv which is used to produce
 * the Type II Plot x Dim Panels plot.
 *
 * */

 val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/CO/extended"

 def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

 def oneTailedExclusive(lt: Double) = if (lt >= 0.95) 1 else 0
 def oneTailedInclusive(lt: Double, eq: Double) = if (lt + eq >= 0.975) 1 else 0
 def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

 var dimMap = Map[(Int, Int), (Int, Int, Int, Int)]()
 val propRange = List(100)

 for (file <- getFiles(path)) {
  println(file.getName)
```

```
 val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

 var hFlag = true
 var hMap = Map[String, Int]()

 for (line <- lines; if (line split ("\t")).length == 13) {
  if (hFlag) {
   hMap = buildMap(line)
   hFlag = false
  } else {
   val currentLine = line split ("\t")
   val n = currentLine(hMap("N")).toInt
   val plots = currentLine(hMap("Plots")).toInt
   val species = currentLine(hMap("Species")).toInt
   val dim = currentLine(hMap("Dim")).toInt
   val proportion = ((n.toDouble / species.toDouble) * 100).toInt
   val dLine = currentLine.map(s => s match {
    case """�""" => Double.PositiveInfinity
    case """?""" => Double.PositiveInfinity
    case """∞""" => Double.PositiveInfinity
    case """-�""" => Double.NegativeInfinity
    case """-?""" => Double.NegativeInfinity
    case """-∞""" => Double.NegativeInfinity
    case _ => s.toDouble
   })

   "id\tcBefore\tobsC\texpC\texpCStd\tlt\teq\tgt\tses\tSpecies\tPlots\tDim\tN"


   if (proportion == 100) {

    if (!dimMap.contains(species, plots)) {
     dimMap += (species, plots) -> {
      (1,
       oneTailedExclusive(dLine(hMap("lt"))),
       oneTailedInclusive(dLine(hMap("lt")), dLine(hMap("eq"))),
       checkSES(dLine(hMap("ses")))
       )
     }
    } else {
     dimMap += (species, plots) -> {
      (dimMap((species, plots))._1 + 1,
       dimMap((species, plots))._2 + oneTailedExclusive(dLine(hMap("lt"))),
       dimMap((species, plots))._3 + oneTailedInclusive(dLine(hMap("lt")), dLine(hMap("eq"))),
       dimMap((species, plots))._4 + checkSES(dLine(hMap("ses")))
       )
      }
     }
    }
   }
  }
 }
```

```
val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/COTypeIIColourExtended.csv"))
f.write("Species,Plots,Count,type,val" + "\n")

println("\nSpecies\tPlots\tCount\ttype\tval")
for (key <- dimMap.keySet.toList.sorted) {
  println(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teq" + "\t" +
dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
     key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tne" + "\t" +
dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
     key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tses" + "\t" +
dimMap(key)._4.toDouble / dimMap(key)._1
  )

  f.write(
    key.toString().replaceAll( """[\(\)]""", "") + "," + dimMap(key)._1 + ",eq" + "," + dimMap(key)._3.toDouble /
dimMap(key)._1 + "\n" +
     key.toString().replaceAll( """[\(\)]""", "") + "," + dimMap(key)._1 + ",ne" + "," + dimMap(key)._2.toDouble /
dimMap(key)._1 + "\n" +
     key.toString().replaceAll( """[\(\)]""", "") + "," + dimMap(key)._1 + ",ses" + "," + dimMap(key)._4.toDouble /
dimMap(key)._1 + "\n"
  )

  f.flush()
}

f.flush()
f.close()

def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.8 NN_TypeIIColour.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}

/*
*
* This script parses the NN Type II sensitivity data and summerises it for R.
* The final product is NTDTypeII_100_Summary.csv which is used to produce
* the Type II Plot x Dim Panels plot.
*
* */

val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/NN"

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator
```

```
def oneTailedExclusive(lt: Double) = if (lt >= 0.95) 1 else 0
def oneTailedInclusive(lt: Double, eq: Double) = if (lt + eq >= 0.975) 1 else 0
def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

def oneTailed(lt: Double, gt: Double) = if (lt >= 0.95) 1 else 0

var dimMap = Map[(Int, Int), (Int, Int, Int, Int, Int, Int, Int)]()
val propRange = List(0, 20, 40, 60, 80, 100)
val spRange = List(5, 10, 15, 20, 25, 30, 35)
val plotRange = List(5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 200)

for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines) {
    if (hFlag) {
      hMap = buildMap(line)
      hFlag = false
    } else {
      val currentLine = line split ("\t")
      val n = currentLine(hMap("N")).toInt
      val plots = currentLine(hMap("plots")).toInt
      val species = currentLine(hMap("species")).toInt
      val dim = plots * species
      val proportion = ((n.toDouble / species.toDouble) * 100).toInt
      val dLine = currentLine.map(s => s match {
        case """<U+FFFD>""" => Double.PositiveInfinity
        case """?""" => Double.PositiveInfinity
        case """∞""" => Double.PositiveInfinity
        case """-<U+FFFD>""" => Double.NegativeInfinity
        case """-?""" => Double.NegativeInfinity
        case """-∞""" => Double.NegativeInfinity
        case _ => s.toDouble
      })

      if (proportion == 100) {

        if (!dimMap.contains(species, plots)) {
          dimMap += (species, plots) -> {
            (1,
              oneTailedExclusive(dLine(hMap("expAITSpLT"))),
              oneTailedInclusive(dLine(hMap("expAITSpLT")), dLine(hMap("expAITSpE"))),
              checkSES(dLine(hMap("expAITSses"))),
              oneTailedExclusive(dLine(hMap("expAWTSpLT"))),
              oneTailedInclusive(dLine(hMap("expAWTSpLT")), dLine(hMap("expAWTSpE"))),
              checkSES(dLine(hMap("expAWTSses")))
              )
```

```
        }
       } else {
        dimMap += (species, plots) -> {
         (dimMap((species, plots))._1 + 1,
           dimMap((species, plots))._2 + oneTailedExclusive(dLine(hMap("expAITSpLT"))),
           dimMap((species, plots))._3 + oneTailedInclusive(dLine(hMap("expAITSpLT")),
dLine(hMap("expAITSpE"))),
           dimMap((species, plots))._4 + checkSES(dLine(hMap("expAITSses"))),
           dimMap((species, plots))._5 + oneTailedExclusive(dLine(hMap("expAWTSpLT"))),
           dimMap((species, plots))._6 + oneTailedInclusive(dLine(hMap("expAWTSpLT")),
dLine(hMap("expAWTSpE"))),
           dimMap((species, plots))._7 + checkSES(dLine(hMap("expAWTSses")))
           )
        }
       }
      }
     }
    }
   }

 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
1/Summarized Data/TraitTypeII_100_Colour.tsv"))
 f.write("Species\tPlots\tCount\ttype\tval" + "\n")

 println("\nSpecies\tPlots\tCount\ttype\tval")
 for (key <- dimMap.keySet.toList.sorted) {
  println(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAITSNTD\t" +
dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAITSNTD\t" +
dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAITSNTD\t" +
dimMap(key)._4.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAWTSNTD\t" +
dimMap(key)._5.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAWTSNTD\t" +
dimMap(key)._6.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAWTSNTD\t" +
dimMap(key)._7.toDouble / dimMap(key)._1
  )

  f.write(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAITSNTD\t" +
dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAITSNTD\t" +
dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAITSNTD\t" +
dimMap(key)._4.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAWTSNTD\t" +
dimMap(key)._5.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAWTSNTD\t" +
dimMap(key)._6.toDouble / dimMap(key)._1 + "\n" +
```

```
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAWTSNTD\t" +
dimMap(key)._7.toDouble / dimMap(key)._1 + "\n"
  )

  f.flush()
 }

 f.flush()
 f.close()

 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

B.2.9 SDNN_TypeIIColour.scala

```
 import io.Source
 import java.io.{FileWriter, BufferedWriter, File}

 /*
 *
 * This script parses the SDNN Type II sensitivity data and summerises it for R.
 * The final product is SDNNTypeII_100_Summary.csv which is used to produce
 * the Type II Plot x Dim Panels plot.
 *
 * */

 val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/SDNN"

 def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

 def oneTailedExclusive(lt: Double) = if (lt >= 0.95) 1 else 0
 def oneTailedInclusive(lt: Double, eq: Double) = if (lt + eq >= 0.975) 1 else 0
 def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

 def oneTailed(lt: Double, gt: Double) = if (lt >= 0.95) 1 else 0

 var dimMap = Map[(Int, Int), (Int, Int, Int, Int, Int, Int, Int)]()
 val propRange = List(0, 20, 40, 60, 80, 100)
 val spRange = List(5, 10, 15, 20, 25, 30, 35)
 val plotRange = List(5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 200)

 for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

  var hFlag = true
  var hMap = Map[String, Int]()
```

```
  for (line <- lines) {
    if (hFlag) {
      hMap = buildMap(line)
      hFlag = false
    } else {
      val currentLine = line split ("\t")
      val n = currentLine(hMap("N")).toInt
      val plots = currentLine(hMap("plots")).toInt
      val species = currentLine(hMap("species")).toInt
      val dim = plots * species
      val proportion = ((n.toDouble / species.toDouble) * 100).toInt
      val dLine = currentLine.map(s => s match {
        case """"""" => Double.PositiveInfinity
        case """?""" => Double.PositiveInfinity
        case """∞""" => Double.PositiveInfinity
        case """-"""" => Double.NegativeInfinity
        case """-?""" => Double.NegativeInfinity
        case """-∞""" => Double.NegativeInfinity
        case _ => s.toDouble
      })

      if (proportion == 100) {

        if (!dimMap.contains(species, plots)) {
          dimMap += (species, plots) -> {
            (1,
              oneTailedExclusive(dLine(hMap("expAITSSDNNpLT"))),
              oneTailedInclusive(dLine(hMap("expAITSSDNNpLT")), dLine(hMap("expAITSSDNNpE"))),
              checkSES(dLine(hMap("expAITSSDNNses"))),
              oneTailedExclusive(dLine(hMap("expAWTSSDNNpLT"))),
              oneTailedInclusive(dLine(hMap("expAWTSSDNNpLT")), dLine(hMap("expAWTSSDNNpE"))),
              checkSES(dLine(hMap("expAWTSSDNNses")))
              )
          }
        } else {
          dimMap += (species, plots) -> {
            (dimMap((species, plots))._1 + 1,
              dimMap((species, plots))._2 + oneTailedExclusive(dLine(hMap("expAITSSDNNpLT"))),
              dimMap((species, plots))._3 + oneTailedInclusive(dLine(hMap("expAITSSDNNpLT")),
dLine(hMap("expAITSSDNNpE"))),
              dimMap((species, plots))._4 + checkSES(dLine(hMap("expAITSSDNNses"))),
              dimMap((species, plots))._5 + oneTailedExclusive(dLine(hMap("expAWTSSDNNpLT"))),
              dimMap((species, plots))._6 + oneTailedInclusive(dLine(hMap("expAWTSSDNNpLT")),
dLine(hMap("expAWTSSDNNpE"))),
              dimMap((species, plots))._7 + checkSES(dLine(hMap("expAWTSSDNNses")))
              )
          }
        }
      }
    }
  }
}
```

```
val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized Data/TraitTypeII_100_Colour.tsv", true))
//f.write("\nSpecies\tPlots\tCount\ttype\tval" + "\n")
//f.write("\n")

println("\nSpecies\tPlots\tCount\ttype\tval")

for (key <- dimMap.keySet.toList.sorted) {
  println(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAITSSDNN\t" + dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAITSSDNN\t" + dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAITSSDNN\t" + dimMap(key)._4.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAWTSSDNN\t" + dimMap(key)._5.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAWTSSDNN\t" + dimMap(key)._6.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAWTSSDNN\t" + dimMap(key)._7.toDouble / dimMap(key)._1
  )

  f.write(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAITSSDNN\t" + dimMap(key)._2.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAITSSDNN\t" + dimMap(key)._3.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAITSSDNN\t" + dimMap(key)._4.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tneAWTSSDNN\t" + dimMap(key)._5.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\teqAWTSSDNN\t" + dimMap(key)._6.toDouble / dimMap(key)._1 + "\n" +
      key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + dimMap(key)._1 + "\tSesAWTSSDNN\t" + dimMap(key)._7.toDouble / dimMap(key)._1 + "\n"
  )

  f.flush()
}

f.flush()
f.close()

def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

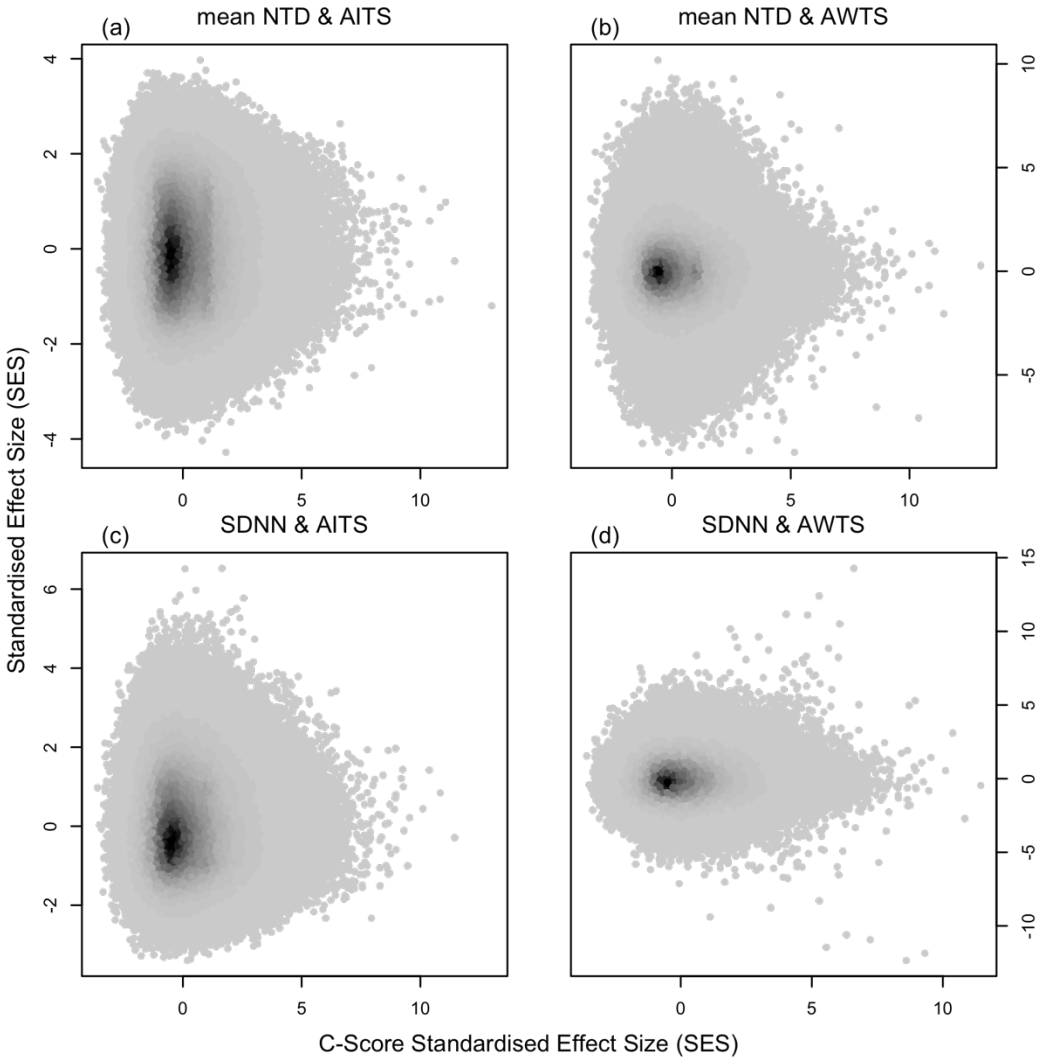

Figure C-1. Standardized effect size (SES) of the limiting similarity null models versus the SES of the co-occurrence null model for the same matrix. All C-Score SES values are skewed to the right. The SES values of the limiting similarity null models shown in panels b, c and d indicate that there is some interaction between C-Score SES and limiting similarity SES values (SDNN & AITS: r = -0.0011, p = 0.0245; mean NN & AITS: r = -0.0001, p = 0.7572; mean NN & AWTS: r = 0.0002, p = 0.6062; SDNN & AWTS: r = -0.0002, p = 0.631).

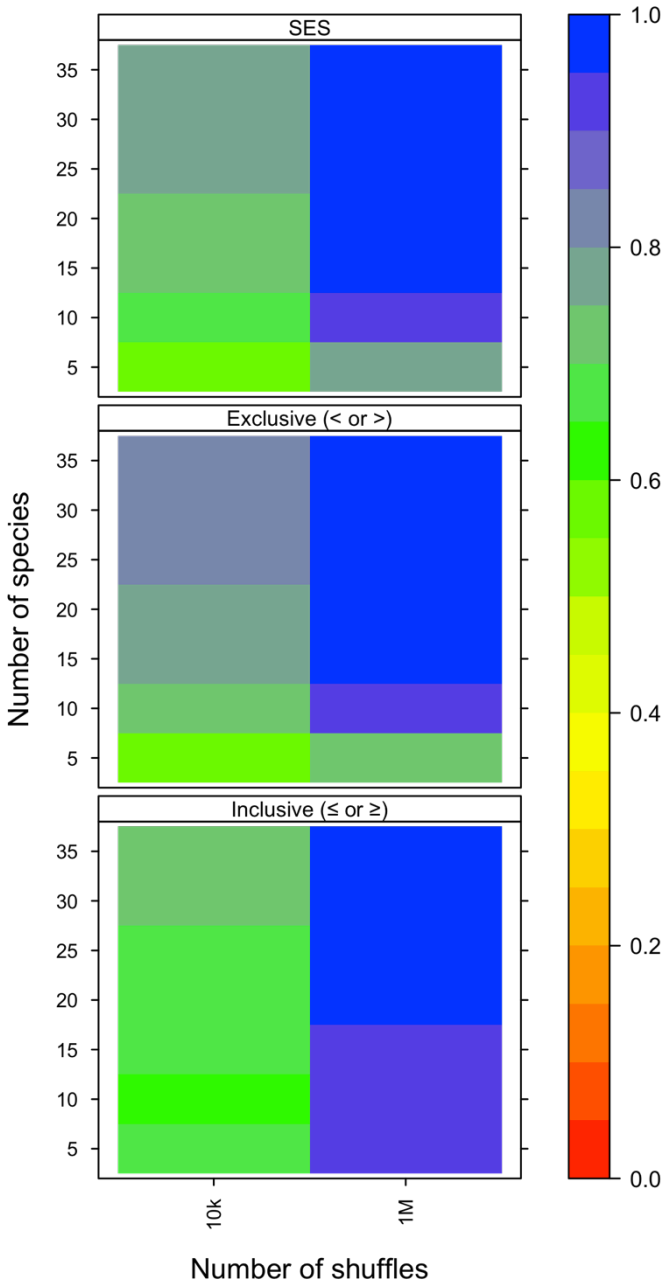

Figure C-2. Type II error rates of the co-occurrence null model test with respect to the number of shuffles used to introduce structure into the matrices. Each panel represents a different criterion for determining the significance of the null model. The colour of each cell indicates the proportion of the 10,000 null models that were significant for that combination of species by plots. Blue cells indicate lower type II error rates and red cells indicate higher type II error rates.

Appendix D Chapter 2.0 R code

D.1.1 'C Score SES x Trait SES.R'

```
rm(list = ls(all = TRUE))

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized Data/TypeIAll.tsv",
 header = TRUE, sep = "\t", na.strings = c("?", "-?", "", "-∞",
        "∞"))

quartz(width = 5.9, height = 6)

#create rowwise plot that is 1 rows by 2 cols in size
par(mfrow = c(2, 2))

#set default font to be 0.6 of default size
par(cex = 0.6)

# set plot margins to 0's and canvas margins to hold titles
par(mar = c(2, 2, 2, 1), oma = c(2, 2, 0, 1))

colors <- densCols(d$AITS_SES_NN, d$SES_C, colramp = colorRampPalette(gray.colors(8,
 start = 0.8, end = 0)))
plot(d$AITS_SES_NN ~ d$SES_C, col = colors, pch = 20, axes = FALSE)
axis(2, col = "black", col.axis = "black") #, at = seq(-4, 4, 1))
axis(1, col = "black", col.axis = "black") #, at = seq(-4, 10, 2))
mtext("(a)", side = 3, line = 0.25, adj = 0.05, cex = 0.8, col = "black")
mtext("mean NN & AITS", side = 3, line = 0.75, col = "black", cex = 0.8)

box(col = "black")

dsub <- subset(d, abs(d$AWTS_SES_NN) < 20)

colors <- densCols(dsub$AWTS_SES_NN ~ dsub$SES_C, colramp = colorRampPalette(gray.colors(8,
 start = 0.8, end = 0)))
plot(dsub$AWTS_SES_NN ~ dsub$SES_C, col = colors, pch = 20, axes = FALSE)

axis(4, col = "black", col.axis = "black") #, at = seq(-4, 4, 1))
axis(1, col = "black", col.axis = "black") #, at = seq(-4, 10, 2))
mtext("(b)", side = 3, line = 0.25, adj = 0.05, cex = 0.8, col = "black")
mtext("mean NN & AWTS", side = 3, line = 0.75, col = "black", cex = 0.8)

box(col = "black")

par(xpd = FALSE)

colors <- densCols(d$AITS_SES_SDNN, d$SES_C, colramp = colorRampPalette(gray.colors(8,
 start = 0.8, end = 0)))
plot(d$AITS_SES_SDNN ~ d$SES_C, col = colors, pch = 20, axes = FALSE)
```

```
axis(2, col = "black", col.axis = "black") #, at = seq(-4, 4, 1))
axis(1, col = "black", col.axis = "black") #, at = seq(-4, 10, 2))
mtext("(c)", side = 3, line = 0.25, adj = 0.05, cex = 0.8, col = "black")
mtext("SDNN & AITS", side = 3, line = 0.75, col = "black", cex = 0.8)

box(col = "black")

dsub <- subset(d, abs(d$AWTS_SES_SDNN) < 20)

colors <- densCols(dsub$AWTS_SES_SDNN, dsub$SES_C, colramp = colorRampPalette(gray.colors(8,
 start = 0.8, end = 0)))
plot(dsub$AWTS_SES_SDNN ~ dsub$SES_C, col = colors, pch = 20, axes = FALSE)

axis(4, col = "black", col.axis = "black") #, at = seq(-4, 4, 1))
axis(1, col = "black", col.axis = "black") #, at = seq(-4, 10, 2))
mtext("(d)", side = 3, line = 0.25, adj = 0.05, cex = 0.8, col = "black")
mtext("SDNN & AWTS", side = 3, line = 0.75, col = "black", cex = 0.8)
#mtext("Standardised Effect Size (SES)", side = 4, line = 2, cex = 0.6, col = "black")
#mtext("C-Score Standardised Effect Size (SES)", side = 1, line = 1.75, cex = 0.6, col = "black")
box(col = "black")

par(xpd = NA)
mtext("Standardised Effect Size (SES)", outer = TRUE, side = 2, line = 0.5,
 cex = 0.8, col = "black")
mtext("C-Score Standardised Effect Size (SES)", outer = TRUE, side = 1,
 line = 0.75, cex = 0.8, col = "black")
```

D.1.2 'C Score x RankAbundance-Evenness

```
rm(list = ls(all = TRUE))

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/cScore x Rank Abundance_Evenness.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA", "?", "-?", "", "-∞",
         "∞"))

str(d)

quartz(width = 2.95, height = 9)

par(mfrow = c(3, 1), mar = c(4, 4, 1, 1), cex = 0.8)

plot(d$cSES ~ d$shanE, pch = "+", data = d, col = "black", xlab = "Shannon's E",
 ylab = "C-Score SES")

plot(d$cSES ~ d$simpE, pch = "+", data = d, col = "black", xlab = "Simpson's E",
 ylab = "C-Score SES")

plot(d$cSES ~ d$ra_m, pch = "+", data = d, col = "black", xlab = "Slope of log transformed\nrank abundance curve",
 ylab = "C-Score SES")
```

```
r_shanE = lm(d$cSES ~ d$shanE)
summary(r_shanE)
r_simpE = lm(d$cSES ~ d$simpE)
summary(r_simpE)
r_ra_m = lm(d$cSES ~ d$ra_m)
summary(r_ra_m)

cor.test(d$cSES, d$shanE)

cor.test(d$cSES, d$simpE)

cor.test(d$cSES, d$ra_m)
```

D.1.3 'CoOccur Type I Contour.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized
 Data/CoOccurTypeIColour.tsv", header = TRUE, sep = "\t")

quartz(width = 4, height = 8)

rgb.palette <- colorRampPalette(c("blue", "green", "yellow", "red"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.75, lines=1.0)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$type
 , levels = c("C_eq", "C_ne", "SES")
 , labels = c("Inclusive (\u2264 or \u2265)",   "Exclusive (< or >)" , "SES")
  ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 3)
 , scales = list(y = list(tck = 0.5
         , alternating = FALSE
         , labels =
```

```
 c("3","4","5","6","7","8","9","10","11","12","13","14","15","16","17","18","19","20","25","30","35","50")
        , cex = 0.65)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "20", "25", "30", "35", "50",
 "75", "100", "150")
        , cex = 0.7))
           , between = list(x = 0.25, y = 0.25)
)
```

D.1.4 'CoOccur Type II Contour.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized
 Data/COTypeIIColour.csv", header = TRUE, sep = ",")

quartz(width = 4, height = 8)

# rgb.palette <- colorRampPalette(c("white", "grey", "black"), space = "rgb")
rgb.palette <- colorRampPalette(c("red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.75, lines=1.0)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$type
 , levels = c("eq", "ne", "ses")
 , labels = c("Inclusive (\u2264 or \u2265)",   "Exclusive (< or >)" , "SES")
  ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 3)
 , scales = list(y = list(tck = 0.5
        , alternating = FALSE
        , labels = c("5","10","15","20","25","30","35")
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
```

```
        , alternating = FALSE
        , labels = c("3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "20", "25", "30", "35", "50")
        , cex = 0.7))
 , between = list(x = 0.25, y = 0.25)
)
```

D.1.5 'CoOccur Type II Extended.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized
 Data/COTypeIIExtendedCombined.csv", header = TRUE, sep = ",")

quartz(width = 4, height = 8)

rgb.palette <- colorRampPalette(c("red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.75, lines=1.0)

levelplot(d$val ~ as.factor(d$Shuffles) * as.factor(d$Species) | ordered(d$type
 , levels = c("eq", "ne", "ses")
 , labels = c("Inclusive (\u2264 or \u2265)",   "Exclusive (< or >)" , "SES")
  ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of shuffles"
 , ylab = "Number of species"
 , layout = c(1, 3)
 , scales = list(y = list(tck = 0.5
        , alternating = FALSE
        , labels = c("5","10","15","20","25","30","35")
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , cex = 0.7))
 , between = list(x = 0.25, y = 0.25)
)
```

D.1.6 'Repeated Matrices x Dim.R'

```
# Mac Path
rm(list = ls(all = TRUE))

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized Data/MatrixRepeats.tsv",
 header = TRUE, sep = "\t", na.strings = c("?",
 "-?", "", "-∞", "∞"))

quartz(width = 5.9, height = 4.5)

par(cex = 0.8)

d <- subset(d, d$Dim <= 60)

plot(d$mean ~ d$Dim, ylim = c(0, 100), xlab = "Community dimension (species x plots)", ylab = "Mean sampling
 occurrence of individual matrices")
for (i in 1:length(d$mean)) {
 segments(d$Dim[i], d$negSTDev[i], d$Dim[i], d$posSTDev[i])
}
```

D.1.7 'Trait Type I Contour.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)


d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized
 Data/TraitTypeIColour.tsv", header = TRUE, sep = "\t")
str(d)

quartz(width = 5.9, height = 8)

rgb.palette <- colorRampPalette(c("blue", "green", "yellow", "red"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.6, lines=1.2)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$type
 , levels = c("AITSntd_eq", "AITSntd_ne", "AITSntd_SES",
                        "AITSsdnn_eq", "AITSsdnn_ne", "AITSsdnn_SES",
                    "AWTSntd_eq", "AWTSntd_ne", "AWTSntd_SES",
                    "AWTSsdnn_eq", "AWTSsdnn_ne", "AWTSsdnn_SES")
 , labels = c("AITS & NN Inclusive",   "AITS & NN Exclusive" , "AITS & NN SES",
                    "AITS & SDNN Inclusive", "AITS & SDNN Exclusive", "AITS & SDNN SES",
                    "AWTS & NN Inclusive",   "AWTS & NN Exclusive" , "AWTS & NN SES",
                    "AWTS & SDNN Inclusive", "AWTS & SDNN Exclusive", "AWTS & SDNN SES")
  ),
 , pretty = FALSE
```

```
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(3, 4)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , labels = c("", "4", "", "6", "", "8", "", "10", "", "12", "", "14", "", "16", "", "18", "","20","","30","","50")
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "", "75", "", "150")
         , cex = 0.7))
 , between = list(x = 0.5, y = 0.5)
)
```

D.1.8 'Trait Type II Contour.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)


d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized
 Data/TraitTypeII_100_Colour.tsv", header = TRUE, sep = "\t")

quartz(width = 5.9, height = 8)

rgb.palette <- colorRampPalette(c("red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.6, lines=1.2)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$type
 , levels = c("eqAITSNTD", "neAITSNTD", "SesAITSNTD",
                          "eqAITSSDNN", "neAITSSDNN", "SesAITSSDNN",
                "eqAWTSNTD", "neAWTSNTD", "SesAWTSNTD",
                "eqAWTSSDNN", "neAWTSSDNN", "SesAWTSSDNN")
 , labels = c("AITS & NN Inclusive",   "AITS & NN Exclusive" , "AITS & NN SES",
                "AITS & SDNN Inclusive", "AITS & SDNN Exclusive", "AITS & SDNN SES",
                "AWTS & NN Inclusive",   "AWTS & NN Exclusive" , "AWTS & NN SES",
                "AWTS & SDNN Inclusive", "AWTS & SDNN Exclusive", "AWTS & SDNN SES")
  ),
```

```
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(3, 4)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , labels = c("5", "10", "15", "20", "25", "30", "35")
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "", "75", "", "150", "")
         , cex = 0.7))
 , between = list(x = 0.5, y = 0.5)
)
```

D.1.9 'logRegression x Trait SES.R'

```
rm(list = ls(all = TRUE))

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 1/Summarized
 Data/TypeIAllOrdered.tsv",
 header = TRUE, sep = "\t", na.strings = c("?", "-?", "", "-∞",
         "∞"))

str(d)

##### Code the data so that I can perform simple Chi-Squared Test

d$c_cat[d$SES_C <= -16] <- "SES <= -16"
d$c_cat[d$SES_C <= -8 & d$SES_C > -16] <- "-16 < SES <= -8"
d$c_cat[d$SES_C <= -4 & d$SES_C > -8] <- "-8 < SES <= -4"
d$c_cat[d$SES_C <= -2 & d$SES_C > -4] <- "-4 < SES <= -2"
d$c_cat[d$SES_C <= -1 & d$SES_C > -2] <- "-2 < SES <= -1"
d$c_cat[d$SES_C <= 0 & d$SES_C > -1] <- "-1 < SES <= -0"
d$c_cat[d$SES_C > 0 & d$SES_C <= 1] <- "0 < SES <= 1"
d$c_cat[d$SES_C > 1 & d$SES_C <= 2] <- "1 < SES <= 2"
d$c_cat[d$SES_C > 2 & d$SES_C <= 4] <- "2 < SES <= 4"
d$c_cat[d$SES_C > 4 & d$SES_C <= 8] <- "4 < SES <= 8"
d$c_cat[d$SES_C > 8 & d$SES_C <= 16] <- "8 < SES <= 16"
d$c_cat[d$SES_C > 16] <- "SES > 16"

# d$sig[abs(d$AITS_SES_NN) > 1.97] <- "sg"
# d$sig[abs(d$AITS_SES_NN) <= 1.97] <- "ns"
```

```
# # Uncomment to seperate into +, - neutral and commment out the previous 2 lines
d$sig[d$AITS_SES_NN < -1.97] <- "-sg"
d$sig[d$AITS_SES_NN >= -1.97 & d$AITS_SES_NN <= 1.96] <- "ns"
d$sig[d$AITS_SES_NN > 1.97] <- "+sg"

d$c_cat <- as.factor(d$c_cat)
d$sig <- as.factor(d$sig)

str(d)

tbl <- table(d$c_cat, d$sig)

Xsq <- chisq.test(tbl)

options(scipen = 100, digits = 0)
Xsq$observed # observed counts (same as M)
Xsq$expected # expected counts under the null

options(scipen = 100, digits = 2)
Xsq$residuals # Pearson residuals
Xsq$stdres
```

Appendix E Chapter 3.0 Scala Code

E.1 Application Code

E.1.1 TypeIICoOccurrence.scala

```
package ca.mikelavender.mscthesis

import ca.mikelavender.nullmodeller.{utils, NullModels, MatrixRandomisation, MatrixStatistics}
import math._
import util.Random
import java.io.FileWriter
import scala.Predef._
import java.text.DecimalFormat
import java.util


object TypeIICoOccurrence {
  def main(args: Array[String]) {
    val runner = new TypeIICoOccurrence

    val usage =
      """
        | Usage: java -jar TypeIICo.jar [--species num] [--plots num] [--X num] [--Y num] [--co_iter num]
        |    [--species num] => number of species you would like in the matrix. (default=100)
        |    [--plots num]   => number of plots you would like in the matrix. (default=100)
        |    [--traits num]  => number of traits to create. (default=10000)
        |    [--X num]       => number of presence/absence matrices to use. (default=1000)
        |    [--Y num]       => number of matrices used for the trait dispersion null model. (default=5000)
        |    [--co_iter num] => number of null matrices to use for each co-occurrence test. (default=5000)
        |    [--permNTD num] => number of permutations to use while maximizing NN value. (default=10000)
        |    [--cpu num]     => number of cpu's (threads) to use when running. (default=1)
        |                       the default is to run as a sequential program (one thread/core).
        |                       This setting really only applies on multi-core systems and is intended
        |                       to allow other programs to share computing power of the system.
        |    [--silent]      => suppresses output to the console window.
        |    usage           => prints this message
        |
        |    Examples: java -jar TypeII.jar --species 50 --plots 50 --X 100 --Y 100
      """.stripMargin

    if (args.contains("usage")) {
      println(usage)
      sys.exit(1)
    }
    val argList = args.toList
    type OptionMap = Map[Symbol, Any]

    def nextOption(map: OptionMap, list: List[String]): OptionMap = {
      def isSwitch(s: String) = (s(0) == '-')
```

```
    list match {
      case Nil => map
      case "--species" :: value :: tail => nextOption(map ++ Map('species -> value.toInt), tail)
      case "--plots" :: value :: tail => nextOption(map ++ Map('plots -> value.toInt), tail)
      case "--X" :: value :: tail => nextOption(map ++ Map('X -> value.toInt), tail)
      case "--Y" :: value :: tail => nextOption(map ++ Map('Y -> value.toInt), tail)
      case "--N" :: value :: tail => nextOption(map ++ Map('N -> value.toInt), tail)
      case "--type" :: value :: tail => nextOption(map ++ Map('type -> value.toString), tail)
      case "--permNTD" :: value :: tail => nextOption(map ++ Map('permNTD -> value.toInt), tail)
      case "--silent" :: value :: tail => nextOption(map ++ Map('silent -> value.toBoolean), tail)
      case string :: opt2 :: tail if isSwitch(opt2) => nextOption(map ++ Map('outfile -> string), list.tail)
      case string :: Nil => nextOption(map ++ Map('outfile -> string), list.tail)
      case option :: tail => println("Unknown option " + option)
      sys.exit(1)
    }
  }
  val options = nextOption(Map(), argList)
  println(options)

  runner.run(options)
 }
}

class TypeIICoOccurrence {
 val factory = new MatrixFactory
 val stats = new MatrixStatistics
 val models = new NullModels
 val shuffle = new MatrixRandomisation
 val config = Config
 val ut = new utils
 var appendFlag = false

 val iter = 5000
 val normalized = false

 val formatter = new DecimalFormat("#0.00")
 val pValue = new DecimalFormat("#0.000000")

 def run(options: Map[Symbol, Any]) {

  config.species = options.getOrElse('species, 10).asInstanceOf[Int]
  config.plots = options.getOrElse('plots, 10).asInstanceOf[Int]
  config.X = options.getOrElse('X, 10000).asInstanceOf[Int]
  config.Y = options.getOrElse('Y, 1).asInstanceOf[Int]
  config.N = options.getOrElse('N, 0).asInstanceOf[Int]
  config.typ = options.getOrElse('type, "nn").asInstanceOf[String]
  config.permNTD = options.getOrElse('permNTD, 10000).asInstanceOf[Int]
  config.silent = options.getOrElse('silent, true).asInstanceOf[Boolean]

  config.outfile = "sp" +
   config.species +
   "p" +
   config.plots +
```

```
  "X" +
  config.X +
  "Y" +
  config.Y +
  "N" +
  config.N +
  "pNTD" +
  config.permNTD +
  "_" +
  Random.nextInt(10000).toString +
  "_TypeIICo.v2.log"

println("Run parameters:")
println("\tSpecies: " + config.species)
println("\tPlots: " + config.plots)
println("\tX: " + config.X)
println("\tY: " + config.Y)
println("\tN: " + config.N)
println("\tpermNTD: " + config.permNTD)

if (config.silent != true) {
  println("Start Time: " + util.Calendar.getInstance().getTime())
  println("id\tcBefore\tobsC\texpC\texpCStd\tlt\teq\tgt\tses\tSpecies\tPlots\tDim\tN")
}
writeToFile("id\tcBefore\tobsC\texpC\texpCStd\tlt\teq\tgt\tses\tSpecies\tPlots\tDim\tN")

var counter = 1
while (counter <= config.X) {
  val matrix = factory.buildMatrix(config.species, config.plots)
  val cBefore = stats.cScore(matrix, config.species, config.plots, normalized)

  var infinityProtection = 0
  while (maximizeCScore(matrix, config.N) != true &
    infinityProtection < config.species * (config.species - 1) &
    !ut.isSwappable(matrix, config.species, config.plots)) {

    infinityProtection += 1
    //do nothing - shuffle the matrix
  }
  if (infinityProtection == config.species * (config.species - 1)) println("FAILED to increase C-Score")
  if (ut.isSwappable(matrix, config.species, config.plots)) {

    val nullModel = models.iSwapNullAnalysisController(matrix, iter, config.throttle, normalized).reverse
    val obsC = nullModel.head
    val expC = mean(nullModel)
    val nullStdDev = stdDev(nullModel)

    if (config.silent != true) {
      print(counter + "\t")
      print(pValue.format(cBefore) + "\t")
      print(pValue.format(obsC) + "\t")
      print(pValue.format(expC) + "\t")
      print(pValue.format(nullStdDev) + "\t")
```

```
      print(pValue.format(nullModel.tail.count(_ < obsC).toDouble / (nullModel.length - 1).toDouble) + "\t")
      print(pValue.format(nullModel.tail.count(_ == obsC).toDouble / (nullModel.length - 1).toDouble) + "\t")
      print(pValue.format(nullModel.tail.count(_ > obsC).toDouble / (nullModel.length - 1).toDouble) + "\t")
      print(pValue.format((obsC - expC) / nullStdDev) + "\t")
      print(config.species + "\t")
      print(config.plots + "\t")
      print(config.species * config.plots + "\t")
      print(config.N + "\n")
     }

     writeToFile(
      counter + "\t" +
       pValue.format(cBefore) + "\t" +
       pValue.format(obsC) + "\t" +
       pValue.format(expC) + "\t" +
       pValue.format(nullStdDev) + "\t" +
       pValue.format(nullModel.tail.count(_ < obsC).toDouble / (nullModel.length - 1).toDouble) + "\t" +
       pValue.format(nullModel.tail.count(_ == obsC).toDouble / (nullModel.length - 1).toDouble) + "\t" +
       pValue.format(nullModel.tail.count(_ > obsC).toDouble / (nullModel.length - 1).toDouble) + "\t" +
       pValue.format((obsC - expC) / nullStdDev) + "\t" +
       config.species + "\t" +
       config.plots + "\t" +
       config.species * config.plots + "\t" +
       config.N
     )

     counter += 1
    } else println("##### Skipped due to infinite loop - unshuffleable")
   }
}

def maximizeCScore(a1: Array[Array[Int]], n: Int) = {

  var changed = false

  if (config.N != 0 & config.N != 1) {

    val sp2maximize = randomSelect(n, a1.indices.toList).sorted

    val subArray = {
     sp2maximize.map(f => a1(f))
    }.toArray
    var maxC = stats.cScore(subArray, n, subArray(0).length, normalized)
    var maxArray = Array.fill(n, a1(0).length)(0)

    var loop = 0

    while (loop <= config.permNTD) {
     loop += 1
     shuffle.coOccFixedEquiProb(subArray)
     val tempC = stats.cScore(subArray, n, subArray(0).length, normalized)
     if (tempC > maxC) {
      maxC = tempC
```

```
      maxArray = subArray.map(_.clone())
      changed = true
     }
    }

    for (i <- 0 to n - 1) {
     a1(sp2maximize(i)) = maxArray(i)
    }

  } else changed = true

  changed
}

def removeAt[A](n: Int, xs: List[A]): (List[A], A) = {
  val (heads, tails) = xs.splitAt(n)
  (heads ::: tails.tail, tails.head)
}

def randomSelect[A](n: Int, xs: List[A]): List[A] = (n, xs) match {
  case (_, Nil) => Nil
  case (0, _) => Nil
  case (_, _) => {
    val x = removeAt(new Random().nextInt(xs.size), xs)
    x._2 :: randomSelect(n - 1, x._1)
  }
}

def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

def stdDev(numbers: List[Double]): Double = {

  var sum: Double = 0.0
  if (numbers.length >= 2) {
    val avg = mean(numbers)
    val factor: Double = 1.0 / (numbers.length.toDouble - 1)
    for (x: Double <- numbers) {
      sum = sum + ((x - avg) * (x - avg))
    }
    sum = sum * factor
  }
  sqrt(sum)
}

def writeToFile(string: String) {
  val out = new FileWriter(config.outfile, appendFlag)
  appendFlag = true
  try {
    out.write(string + "\n")
  } finally {
    out.close()
  }
}
```

```
}
```

E.1.2 TypeIISensitivity.scala

```
package ca.mikelavender.mscthesis

import util.Random
import ca.mikelavender.nullmodeller.{MatrixRandomisation, MatrixStatistics}
import collection.mutable.ArrayBuffer
import math._
import java.text.DecimalFormat
import scala.collection.immutable.IndexedSeq
import java.io.FileWriter

object TypeIISensitivity {
  def main(args: Array[String]) {

    val usage =
      """
        | Usage: java -jar TypeII.jar [--species num] [--plots num] [--X num] [--Y num] [--co_iter num]
        |    [--species num] => number of species you would like in the matrix. (default=100)
        |    [--plots num]   => number of plots you would like in the matrix. (default=100)
        |    [--traits num]  => number of traits to create. (default=10000)
        |    [--X num]       => number of presence/absence matrices to use. (default=1000)
        |    [--Y num]       => number of matrices used for the trait dispersion null model. (default=5000)
        |    [--co_iter num] => number of null matrices to use for each co-occurrence test. (default=5000)
        |    [--permNTD num] => number of permutations to use while maximizing NTD value. (default=10000)
        |    [--cpu num]     => number of cpu's (threads) to use when running. (default=1)
        |                       the default is to run as a sequential program (one thread/core).
        |                       This setting really only applies on multi-core systems and is intended
        |                       to allow other programs to share computing power of the system.
        |    [--silent]      => suppresses output to the console window.
        |    usage           => prints this message
        |
        |    Examples: java -jar TypeII.jar --species 50 --plots 50 --X 100 --Y 100
      """.stripMargin

    val runner = new TypeIISensitivity

    if (args.contains("usage")) {
      println(usage)
      sys.exit(1)
    }
    val argList = args.toList
    type OptionMap = Map[Symbol, Any]

    def nextOption(map: OptionMap, list: List[String]): OptionMap = {
      def isSwitch(s: String) = (s(0) == '-')
      list match {
        case Nil => map
        case "--species" :: value :: tail => nextOption(map ++ Map('species -> value.toInt), tail)
```

```
      case "--plots" :: value :: tail => nextOption(map ++ Map('plots -> value.toInt), tail)
      case "--X" :: value :: tail => nextOption(map ++ Map('X -> value.toInt), tail)
      case "--Y" :: value :: tail => nextOption(map ++ Map('Y -> value.toInt), tail)
      case "--N" :: value :: tail => nextOption(map ++ Map('N -> value.toInt), tail)
      case "--type" :: value :: tail => nextOption(map ++ Map('type -> value.toString), tail)
      case "--permNTD" :: value :: tail => nextOption(map ++ Map('permNTD -> value.toInt), tail)
      case "--silent" :: value :: tail => nextOption(map ++ Map('silent -> value.toBoolean), tail)
      case string :: opt2 :: tail if isSwitch(opt2) => nextOption(map ++ Map('outfile -> string), list.tail)
      case string :: Nil => nextOption(map ++ Map('outfile -> string), list.tail)
      case option :: tail => println("Unknown option " + option)
       sys.exit(1)
     }
    }
    val options = nextOption(Map(), argList)
    println(options)
    runner.run(options)
  }
}

class TypeIISensitivity {
  val config = Config

  val mtxFactory = new MatrixFactory
  val mtxStats = new MatrixStatistics
  val shuffler = new MatrixRandomisation
  val models = new NullModelsLocal
  var appendFlag = false

  val formatter = new DecimalFormat("#0.00")
  val pValue = new DecimalFormat("#0.0000")

  def run(options: Map[Symbol, Any]) {

    config.species = options.getOrElse('species, 10).asInstanceOf[Int]
    config.plots = options.getOrElse('plots, 10).asInstanceOf[Int]
    config.X = options.getOrElse('X, 10000).asInstanceOf[Int]
    config.Y = options.getOrElse('Y, 1).asInstanceOf[Int]
    config.N = options.getOrElse('N, 0).asInstanceOf[Int]
    config.typ = options.getOrElse('type, "ntd").asInstanceOf[String]
    config.permNTD = options.getOrElse('permNTD, 10000).asInstanceOf[Int]
    config.silent = options.getOrElse('silent, true).asInstanceOf[Boolean]

    config.outfile = "sp" +
     config.species +
     "p" +
     config.plots +
     "X" +
     config.X +
     "Y" +
     config.X +
     "N" +
     config.N +
     "pNTD" +
```

```
      config.permNTD +
      "T" +
      config.typ +
      "_" +
      Random.nextInt(10000).toString +
      "_TypeIISen.v5.log"

    println("Run parameters:")
    println("\tSpecies: " + config.species)
    println("\tPlots: " + config.plots)
    println("\tX: " + config.X)
    println("\tY: " + config.Y)
    println("\tN: " + config.N)
    println("\tType: " + config.typ)
    println("\tpermNTD: " + config.permNTD)


    /**
     * 1. Create a PA matrix of dimension Sp x Pl.
     * 2. Create an Array of Sp traits.
     * 3. Determine all -ve co-occurring species.
     * 4. Pick N species (not -ve co-occurring) and maximize NTD - full permutations/combinations.
     * 5. Assign trait values to remaining species randomly.
     * 6. Calculate NTD
     * 7. Do null models (AITS & AWTS)
     * 8. Repeat 5 - 7 for all combinations of N species
     * 9. Increment N
     * 10. Repeat 5 - 9 for 2 <= N <= (trait count - -ve co-occurring count)
     * 11. Repeat 1 - 10 for X matrices of dimension Sp x Pl
     */

    val hString = {
      if (config.typ == "ntd") {
        "id\tobservedC\tobsNTD\t" +
          "expAITSNTD\texpAITSNTDStDev\texpAITSNTDpLT\texpAITSNTDpE\texpAITSNTDpGT\texpAITSNTDses\t"
+

"expAWTSNTD\texpAWTSNTDStDev\texpAWTSNTDpLT\texpAWTSNTDpE\texpAWTSNTDpGT\texpAWTSNTDses
\t" +
          "species\tplots\tdim\tN"
      } else {
        "id\tobservedC\tobsSDNN\t" +

"expAITSSDNN\texpAITSSDNNStDev\texpAITSSDNNpLT\texpAITSSDNNpE\texpAITSSDNNpGT\texpAITSSDNNses
\t" +

"expAWTSSDNN\texpAWTSSDNNStDev\texpAWTSSDNNpLT\texpAWTSSDNNpE\texpAWTSSDNNpGT\texpAWT
SSDNNses\t" +
          "species\tplots\tdim\tN"
      }
    }

    if (!config.silent) println(hString)
```

```
    writeToFile(hString)

    var mLoop = 0
    while (mLoop < config.X) {
     var pa = Array.empty[Array[Int]]
     var nonNegOccurringSp = List[Int]()

     do {
      // generate PA matrix
      pa = mtxFactory.buildMatrix(config.species, config.plots)

      // get -ve occurring species
      val negOccurringSp = getAllPairwiseVeech(pa).filter(p => p._3._2 + p._3._1 <= 0.05).groupBy(f =>
f._1).map(_._1).toSet

      // get non -ve occurring species
      nonNegOccurringSp = (0 to config.species - 1).toSet.diff(negOccurringSp).toList

      // the logic here is if N = 0 or one then we don't care about +/- co-occurrence - it is just random
     } while (config.N > 1 & nonNegOccurringSp.length < config.N)
     // generate trait values
     val traits = Array.fill(config.species)(math.abs((Random.nextDouble() + 0.001) * 10000).toInt / 100d)

     val maximal = {
      if (config.typ == "ntd")
       getMaximalNTD(nonNegOccurringSp, traits, pa)
      else getMinimalSDNN(nonNegOccurringSp, traits, pa)
     }

     // return a Map of species -> traits
     val allAssigned: Array[Array[Double]] = assignTraitsToMatrix(pa, traits, maximal)

     //     println("Max ********************")
     //     println(maximal)
     //     println("Traits ********************")
     //     println(traits.mkString(", "))
     //     println("Before ********************")
     //     println(pa.deep.mkString("\n"))
     //     println("After ********************")
     //     println(allAssigned.deep.mkString("\n"))

     someMethod(allAssigned, pa, traits, config.species, config.plots, config.N, mLoop)

     mLoop += 1
    }
   }

  //todo: test this
  def assignTraitsToMatrix(pa: Array[Array[Int]], traits: Array[Double], maximal: Map[Int, Double]):
Array[Array[Double]] = {
    val allAssigned = {
     if (config.N > 1) {
      assignTraits(pa, getRemainders(traits, maximal, pa.length) ++ maximal)
```

```
      } else
        assignTraits(pa, traits)
      }
      allAssigned
    }

    def getMaximalNTD(nonNegOccurringSp: List[Int], traits: Array[Double], pa: Array[Array[Int]]): Map[Int, Double]
= {

      var maximal: Map[Int, Double] = Map()
      var maxNTD = 0d

      // start maximise mNTD loop (100000 times or more)
      var loop = 0
      while (loop < config.permNTD & config.N > 1) {

        // randomly choose N species indices from non -ve occurring
        val choosenSpecies = randomSelect(config.N, nonNegOccurringSp)

        // randomly choose N trait values
        val choosenTraits = randomSelect(config.N, traits.toList)

        // create trait sub matrix and store species -> trait Map
        var map: Map[Int, Double] = Map()
        var traitCounter = 0
        val traitSubMatrix = Array.fill(pa.length, pa(0).length)(0d)
        for (r <- choosenSpecies) {
          traitSubMatrix(r) = pa(r).map(_ * choosenTraits(traitCounter))
          map += (r -> choosenTraits(traitCounter))
          traitCounter += 1
        }

        // calc mNTD
        val currentNTD = mtxStats.meanAndStdDevNTD(traitSubMatrix
          .filter(p => p.sum != 0)
          .transpose
          .filter(p => p.sum != 0)
          .transpose)

        // check to see if it is larger than the last or 0
        if (currentNTD._1 > maxNTD | maxNTD == 0) {
          maxNTD = currentNTD._1
          maximal = map
        }

        loop += 1
      }
      maximal
    }

    def getMinimalSDNN(nonNegOccurringSp: List[Int], traits: Array[Double], pa: Array[Array[Int]]): Map[Int, Double]
= {
```

```
    var maximal: Map[Int, Double] = Map()
      var maxNTD = Double.MaxValue
//    var maxNTD = 0d

    // start maximise mNTD loop (100000 times or more)
    var loop = 0
    while (loop < config.permNTD & config.N > 1) {

     // randomly choose N species indices from non -ve occurring
     val choosenSpecies = randomSelect(config.N, nonNegOccurringSp)

     // randomly choose N trait values
     val choosenTraits = randomSelect(config.N, traits.toList)

     // create trait sub matrix and store species -> trait Map
     var map: Map[Int, Double] = Map()
     var traitCounter = 0
     val traitSubMatrix = Array.fill(pa.length, pa(0).length)(0d)
     for (r <- choosenSpecies) {
      traitSubMatrix(r) = pa(r).map(_ * choosenTraits(traitCounter))
      map += (r -> choosenTraits(traitCounter))
      traitCounter += 1
     }

     // calc mNTD
     val currentNTD = mtxStats.meanAndStdDevNTD(traitSubMatrix
      .filter(p => p.sum != 0)
      .transpose
      .filter(p => p.sum != 0)
      .transpose)

     //todo: this should be changed to < as Brandon pointed out it should be evenly spaced under LS. i.e. smaller SD
     // check to see if it is larger than the last or 0
         if (currentNTD._2 < maxNTD | maxNTD == Double.MaxValue) {
//     if (currentNTD._2 > maxNTD | maxNTD == 0) {
      maxNTD = currentNTD._2 // ._2 is the stDev values versus _.1 which is mean values
      maximal = map
     }

     loop += 1
    }
    maximal
   }

   def assignTraits(paMatrix: Array[Array[Int]], traitMap: Map[Int, Double]): Array[Array[Double]] = {
    val tMatrix = Array.fill(paMatrix.length, paMatrix(0).length)(0d)
    for (i <- 0 to paMatrix.length - 1) tMatrix(i) = paMatrix(i).map(_ * traitMap(i))
    tMatrix
   }

   def assignTraits(paMatrix: Array[Array[Int]], traitArray: Array[Double]): Array[Array[Double]] = {
    val tMatrix = Array.fill(paMatrix.length, paMatrix(0).length)(0d)
    for (i <- 0 to paMatrix.length - 1) tMatrix(i) = paMatrix(i).map(_ * traitArray(i))
```

```
    tMatrix
  }

  def someMethod(theMatrix: Array[Array[Double]], pa: Array[Array[Int]], traits: Array[Double], species: Int, plots:
Int, n: Int, loop: Int) {

    // now i need to do all the null model tests
    val observedC = formatter.format(mtxStats.cScore(pa, species, plots, true))
    val observed = mtxStats.meanAndStdDevNTD(theMatrix)
    val aits = models.nullModelAITS(pa, traits)
    val awts = models.nullModelAWTS(pa, traits)

    val obsMean = observed._1
    val obsSDNN = observed._2

    val expAITS = mean(aits._1)
    val expAITSStDev = stdDev(aits._1)
    val expAITSpLT = pValue.format(aits._1.count(_ < obsMean).toDouble / aits._1.length.toDouble)
    val expAITSpE = pValue.format(aits._1.count(_ == obsMean).toDouble / aits._1.length.toDouble)
    val expAITSpGT = pValue.format(aits._1.count(_ > obsMean).toDouble / aits._1.length.toDouble)
    val expAITSses = pValue.format((obsMean - expAITS) / expAITSStDev)

    val expAWTS = mean(awts._1)
    val expAWTSStDev = stdDev(awts._1)
    val expAWTSpLT = pValue.format(awts._1.count(_ < obsMean).toDouble / awts._1.length.toDouble)
    val expAWTSpE = pValue.format(awts._1.count(_ == obsMean).toDouble / awts._1.length.toDouble)
    val expAWTSpGT = pValue.format(awts._1.count(_ > obsMean).toDouble / awts._1.length.toDouble)
    val expAWTSses = pValue.format((obsMean - expAWTS) / expAWTSStDev)


    val expAITSSDNN = mean(aits._2)
    val expAITSSDNNStDev = stdDev(aits._2)
    val expAITSSDNNpLT = pValue.format(aits._2.count(_ < obsSDNN).toDouble / aits._2.length.toDouble)
    val expAITSSDNNpE = pValue.format(aits._2.count(_ == obsSDNN).toDouble / aits._2.length.toDouble)
    val expAITSSDNNpGT = pValue.format(aits._2.count(_ > obsSDNN).toDouble / aits._2.length.toDouble)
    val expAITSSDNNses = pValue.format((obsSDNN - expAITSSDNN) / expAITSSDNNStDev)

    val expAWTSSDNN = mean(awts._2)
    val expAWTSSDNNStDev = stdDev(awts._2)
    val expAWTSSDNNpLT = pValue.format(awts._2.count(_ < obsSDNN).toDouble / awts._2.length.toDouble)
    val expAWTSSDNNpE = pValue.format(awts._2.count(_ == obsSDNN).toDouble / awts._2.length.toDouble)
    val expAWTSSDNNpGT = pValue.format(awts._2.count(_ > obsSDNN).toDouble / awts._2.length.toDouble)
    val expAWTSSDNNses = pValue.format((obsSDNN - expAWTSSDNN) / expAWTSSDNNStDev)

    val oString = {
      if (config.typ == "ntd") {
        loop + 1 + "\t" + observedC + "\t" + formatter.format(obsMean) + "\t" +
          formatter.format(expAITS) + "\t" + formatter.format(expAITSStDev) + "\t" + expAITSpLT + "\t" + expAITSpE
+ "\t" + expAITSpGT + "\t" + expAITSses + "\t" +
          formatter.format(expAWTS) + "\t" + formatter.format(expAWTSStDev) + "\t" + expAWTSpLT + "\t" +
expAWTSpE + "\t" + expAWTSpGT + "\t" + expAWTSses + "\t" +
          species + "\t" + plots + "\t" + species * plots + "\t" + n
      } else {
```

```
      loop + 1 + "\t" + observedC + "\t" + formatter.format(obsSDNN) + "\t" +
       formatter.format(expAITSSDNN) + "\t" + formatter.format(expAITSSDNNStDev) + "\t" + expAITSSDNNpLT +
"\t" + expAITSSDNNpE + "\t" + expAITSSDNNpGT + "\t" + expAITSSDNNses + "\t" +
       formatter.format(expAWTSSDNN) + "\t" + formatter.format(expAWTSSDNNStDev) + "\t" +
expAWTSSDNNpLT + "\t" + expAWTSSDNNpE + "\t" + expAWTSSDNNpGT + "\t" + expAWTSSDNNses + "\t" +
       species + "\t" + plots + "\t" + species * plots + "\t" + n
     }
    }

    if (!config.silent) println(oString)
    writeToFile(oString)
  }


  def getRemainders(allTraits: Array[Double], traitMap: Map[Int, Double], paSize: Int): Map[Int, Double] = {
    val traits = traitMap.map(_._2).toArray
    val species = traitMap.map(_._1).toArray
    val unusedTraits = allTraits.diff(traits)
    val traitlessSpecies = (0 to paSize - 1).toArray.diff(species)
    if (unusedTraits.size != traitlessSpecies.size) {
      sys.error("Oh Shit!!!\n" +
        "all traits: " + allTraits.sorted.mkString(", ") + "\n" +
        "unused traits: " + unusedTraits.toArray.sorted.mkString(", ") + "\n" +
        "species: " + traitMap.map(_._1).toArray.sorted.mkString(", ") + "\n" +
        "traitless species: " + traitlessSpecies.toArray.sorted.mkString(", ") + "\n")
    } else
      Map(shuffle(traitlessSpecies.toArray).zip(shuffle(unusedTraits.toArray)).toArray: _*)
  }


  def canMaximizeAll(paMatrix: Array[Array[Int]], speciesIndices: List[Int]): Boolean = {
    val testMatrix = for (i <- speciesIndices) yield paMatrix(i).clone()
    val test = testMatrix.transpose.map(_.sum)
    test.filter(_ > 1).length > 0
  }

  def applyMapToMatrix(map: Map[Int, Double], paMatrix: Array[Array[Int]]): Array[Array[Double]] = {
    val traitSubMatrix = Array.fill(paMatrix.length, paMatrix(0).length)(0d)
    for (key <- map.keysIterator) traitSubMatrix(key) = paMatrix(key).map(_ * map(key))

    traitSubMatrix
  }


  def getAllPairwiseVeech(matrix: Array[Array[Int]]): IndexedSeq[(Int, Int, (Double, Double, Double))] = {
    for (x <- 0 to matrix.length - 1; y <- 0 to matrix.length - 1)
    yield (x, y, mtxStats.veech(matrix(x), matrix(y)))
  }

  def kissPairwise(matrix: Array[Array[Int]]) = {
    def multiplyRows(a: Array[Int], b: Array[Int], sizeHint: Int): Int = {
      val l: Array[Int] = new Array(sizeHint)
      var i = 0
```

```
    while (i < sizeHint) {
      l(i) = a(i) * b(i)
      i += 1
    }
    l.sum
  }

  for (x <- 0 to matrix.length - 1; y <- 0 to matrix.length - 1)
  yield (x, y, multiplyRows(matrix(x), matrix(y), matrix(x).length))
}

def removeAt[A](n: Int, xs: List[A]): (List[A], A) = {
  val (heads, tails) = xs.splitAt(n)
  (heads ::: tails.tail, tails.head)
}

def randomSelect[A](n: Int, xs: List[A]): List[A] = (n, xs) match {
  case (_, Nil) => Nil
  case (0, _) => Nil
  case (_, _) => {
    val x = removeAt(new Random().nextInt(xs.size), xs)
    x._2 :: randomSelect(n - 1, x._1)
  }
}

def getSpeciesSet(paMatrix: Array[Array[Int]], negMap: Map[Int, Double]): Set[Int] = {
  negMap.isEmpty match {
    case true => (0 to paMatrix.length - 1).toSet
    case false => (0 to paMatrix.length - 1).toSet -- negMap.map(_._1).toSet
    case _ => sys.error("WE HAVE A PROBLEM HUSTON!! - There is no set of species to assign traits to!")
  }
}

def shuffle[T](array: Array[T]) = {
  val buf = new ArrayBuffer[T] ++= array

  def swap(i1: Int, i2: Int) {
    val tmp = buf(i1)
    buf(i1) = buf(i2)
    buf(i2) = tmp
  }

  for (n <- buf.length to 2 by -1) {
    val k = Random.nextInt(n)
    swap(n - 1, k)
  }
  buf
}

def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

def stdDev(numbers: List[Double]): Double = {
```

```
    var sum: Double = 0.0
    if (numbers.length >= 2) {
      val avg = mean(numbers)
      val factor: Double = 1.0 / (numbers.length.toDouble - 1)
      for (x: Double <- numbers) {
        sum = sum + ((x - avg) * (x - avg))
      }
      sum = sum * factor
    }
    sqrt(sum)
  }

  def writeToFile(string: String) {
    val out = new FileWriter(config.outfile, appendFlag)
    appendFlag = true
    try {
      out.write(string + "\n")
    } finally {
      out.close()
    }
  }

}
```

E.1.3 Config.scala

```
package ca.mikelavender.mscthesis


 object Config {

  var normalized = false
  var throttle = 1
  var iter = 5000

  var species = 100
  var plots = 100
  var traitsCount = 100
  var permNTD = 10000
  var X = 1000
  var Y = 5000
  var silent = false
  var outfile = ""
  var percentStructure = .2
  var N = 0
  var typ = "nn"


 }
```

E.1.4 FileTools.scala

```
package ca.mikelavender.mscthesis

import io.Source
import scala.Array._

class FileTools {

  // Scans a CSV file to try and determine the number of rows and columns.
  def scan(file: String, delimiter: String) = {
    var sites_col = 0
    var species_row = 0
    val source = Source.fromFile(file)

    // figure out the number columns in the file given a delimiter
    sites_col = source.getLines().next().split(delimiter).length

    // now get the number of rows but add one on since we used one up figuring out the col's
    species_row = source.getLines().filter(s => s.length > 1).size + 1

    // Done - close the file
    source.close()
    (sites_col, species_row)
  }

  // Reads a file and stores it in a Matrix object.
  def read(file: String, hasHeader: Boolean = false, hasRowLabels: Boolean = false, delimiter: String = ",") = {
    val matrixDimensions = scan(file, delimiter)
    var sites_col = matrixDimensions._1
    var species_row = matrixDimensions._2
    var x = 0

    // Configure the Array
    (hasHeader, hasRowLabels) match {
      case (true, true) => sites_col = sites_col - 1; species_row = species_row - 1
      case (true, false) => species_row = species_row - 1
      case (false, true) => sites_col = sites_col - 1
      case (_, _) => None
    }
    if (hasHeader) print("File has header row. ") else print("File has no header row. ")
    if (hasRowLabels) println("File has row labels.") else println("File has no row labels.")
    println("\tConfigured " + species_row + " x " + sites_col + " matrix.")
    val matrix: Array[Array[Int]] = ofDim(species_row, sites_col)

    val source = Source.fromFile(file)
    var header: IndexedSeq[String] = IndexedSeq()
    var labels: IndexedSeq[String] = IndexedSeq()
    for (line <- source.getLines()) {
      if (x == 0 & hasHeader) {
        header = line.split(delimiter)
      } else {
        val parsed = parse(line, delimiter, hasRowLabels)
```

```
      matrix(x) = parsed._2
      if (hasRowLabels) labels = labels :+ parsed._1
      x += 1
     }
    }
    source.close()
    (matrix, header, labels)
  }

  def parse(line: String, delimiter: String, hasLabels: Boolean) = {
    val data = line.split(delimiter)
    hasLabels match {
      case true => (data(0), data.drop(1).map(_.trim.toInt))
      case _ => ("", data.map(_.trim.toInt))
    }
  }
}
```

E.1.5 MatrixFactory.scala

```
package ca.mikelavender.mscthesis

import util.Random
import ca.mikelavender.nullmodeller.utils

class MatrixFactory {
  private val utils = new utils

  /*Builds a random matrix using row and column constraints.
  * Row constraints - abundance for each row sampled from log-normal distribution
  * Column constraints - proportion for each column sampled from uniform distribution
  * For methods see:
  * Ulrich, W., & Gotelli, N. J. (2010). Null model analysis of species associations using abundance data. Ecology, 91(11), 3384–3397. doi:10.1890/09-2157.1
  * Gotelli, N. J. (2000). Null Model Analysis of Species Co-Occurrence Patterns. Ecology, 81(9), 2606. doi:10.2307/177478
  * */

  //todo: I don't need to constrain the random matrix to proportion only the null matrices which are constrained by the original random matrix proportions.
  def buildMatrix(species: Int, plots: Int): Array[Array[Int]] = buildMatrix(species, plots, "logNormal")

  def buildMatrix(species: Int, plots: Int, speciesConstraint: String): Array[Array[Int]] = {

    def go: Array[Array[Int]] = {
      val spConstArray = getSpeciesConstraints(species, plots, speciesConstraint)
      val matrix = Array.fill(species, plots)(0)
      for (spIndex <- 0 to species - 1)
        matrix(spIndex) = buildConstrainedRow(spIndex, plots, spConstArray)
      if (constrainColumns(matrix) & utils.isSwappable(matrix, species, plots)) matrix else go
    }
```

```
    go
  }

  def buildConstrainedRow(rowIndex: Int, plots: Int, speciesConstraints: Array[Int]): Array[Int] = {
    var workingRow = Array.fill(plots)(if (Random.nextDouble() <= speciesConstraints(rowIndex).toDouble / plots) 1
else 0)

    def constrainRow {
      workingRow.sum.compare(speciesConstraints(rowIndex)) match {
        case -1 => {
          val zeros = workingRow.zipWithIndex.filter(_._1 == 0).map(_._2)
          workingRow = workingRow.updated(zeros(Random.nextInt(zeros.size)), 1)
          constrainRow
        }
        case 1 => {
          val ones = workingRow.zipWithIndex.filter(_._1 == 1).map(_._2)
          workingRow = workingRow.updated(ones(Random.nextInt(ones.size)), 0)
          constrainRow
        }
        case 0 => // Nothing to do. the row meets constraints
      }
    }
    constrainRow

    workingRow
  }


  // creates a single row for the matrix using column proportion as the constraint
  def buildRow(sites: Int, colProportions: Array[Double]) = {
    for (c <- 0 to sites - 1)
    yield {
      if (Random.nextDouble() <= colProportions(sites - 1)) 1
      else 0
    }
  }

  // Makes sure that each plot has at least one species in it.
  def constrainColumns(matrix: Array[Array[Int]]): Boolean = {
    if (matrix.
      transpose.
      map(_.sum).
      zipWithIndex.
      filter(_._1 == 0).
      map(_._2).length > 0) false
    else true
  }

  def getSpeciesConstraints(species: Int, plots: Int, speciesConstraint: String): Array[Int] = {
    speciesConstraint match {
      case "logNormal" => getArrayOfNonZeroLogNormals(species, plots)
      //todo: need to implement these in helpers class
```

```
    case "normal" => sys.error("Normal distribution not implemented yet")
    case "uniform" => sys.error("Uniform distribution not implemented yet")
    case _ => sys.error("ABORT: No valid distribution given for species abundances")
   }
  }

  private def getNextLogNormal(a: Double) = math.exp(Random.nextGaussian() / (2 * a))

  def getNextLogNormal(maxVal: Int, a: Double = 0.2): Int = {
   if (maxVal <= 0) sys.error("ABORT: Invalid maximum value for log-normal distribution given")
   var l = 0
   while (l == 0) {
    val nextInt = getNextLogNormal(a).toInt
    if (nextInt != 0 & nextInt <= maxVal) l = nextInt
   }
   l
  }

  def getArrayOfNonZeroLogNormals(species: Int, plots: Int = 0, a: Double = 0.2): Array[Int] = {
   var l: List[Int] = Nil
   while (l.size < species) {
    val nextInt = getNextLogNormal(a).toInt
    if (nextInt != 0 & plots == 0) l = nextInt :: l
    else if (plots > 0 & nextInt <= plots & nextInt != 0) l = nextInt :: l
   }
   l.toArray
  }
 }
```

E.1.6 NullModelsLocal.scala

```
package ca.mikelavender.mscthesis

 import ca.mikelavender.nullmodeller.{MatrixRandomisation, MatrixStatistics}


 class NullModelsLocal {
  val mtxStats = new MatrixStatistics
  val shuffler = new MatrixRandomisation

  def nullModelAITS(matrix: Array[Array[Int]], traits: Array[Double], loops: Int = 5000): (List[Double], List[Double])
= {
   var meanNTD_AITS: List[Double] = Nil
   var meanSDNN_AITS: List[Double] = Nil

   var loop = 0
   while (loop < loops) {
    val statsAITS = mtxStats.meanAndStdDevNTD(shuffler.traitAITS(matrix, traits))
    meanNTD_AITS = statsAITS._1 :: meanNTD_AITS
    meanSDNN_AITS = statsAITS._2 :: meanSDNN_AITS
    loop += 1
```

```
    }
    (meanNTD_AITS, meanSDNN_AITS)
  }

  def nullModelAWTS(matrix: Array[Array[Int]], traits: Array[Double], loops: Int = 5000): (List[Double], List[Double])
= {
    var meanNTD_AWTS: List[Double] = Nil
    var meanSDNN_AWTS: List[Double] = Nil

    var loop = 0
    while (loop < loops) {
      val statsAWTS = mtxStats.meanAndStdDevNTD(shuffler.traitAWTS(matrix, traits))
      meanNTD_AWTS = statsAWTS._1 :: meanNTD_AWTS
      meanSDNN_AWTS = statsAWTS._2 :: meanSDNN_AWTS
      loop += 1
    }
    (meanNTD_AWTS, meanSDNN_AWTS)
  }

}
```

E.1.7 TraitAssignment.scala

```
package ca.mikelavender.mscthesis

import ca.mikelavender.nullmodeller.MatrixStatistics
import annotation.tailrec
import util.Random
import collection.mutable.ArrayBuffer
import scala.collection.immutable

class TraitAssignment {

  val mtxStats = new MatrixStatistics
  val config = Config

  def traitAssignment(paMatrix: Array[Array[Int]], tValues: Array[Double]): Array[Array[Double]] = {
    /*
    // 1. Determine all +ve co-occurring groups of species
    // 2. Determine all -ve co-occurring groups of species
    // 3a. If no -ve and no +ve co-occurring species then assign traits randomly
    // 3b. If no -ve co-occurring species then pool the +ve groups and assign traits to maximize NN
    // 3c. If there is -ve co-occurrence, pool groups with non -ve relationships and assign traits to the largest group
and maximize NN
    // 4. Assign remaining trait values randomly to the remaining species
    // 5. Return a Matrix of Doubles
    // NOTE: I also need to keep track of the number of matrices with +ve co-occurrence. The Limiting Similarity test
should match.
    */

    val allVals = getAllPairwiseVeech(paMatrix)
```

```
    def coOccur(n: Int): Set[Int] = {
      val av = allVals
        .filter(p => p._1 == n & p._1 != p._2 & p._3._2 + p._3._3 <= 0.05)
        .map(f => Set(f._1, f._2))

      if (av.isEmpty) {
        Set[Int]()
      } else
        av.reduceLeft(_ ++ _)
    }

    val negCoOccur = allVals.filter(p => p._3._2 + p._3._1 <= 0.05).groupBy(f => f._1)

    var i = 0
    var coList: List[Set[Int]] = Nil
    while (i < paMatrix.length) {
      coList = coOccur(i) :: coList
      i += 1
    }

    val rowIndices = getSpeciesSet(paMatrix, negCoOccur, coList).toIndexedSeq
    val traitSubMatrix = Array.fill(paMatrix.length, paMatrix(0).length)(0d)

    var maximal: Map[Int, Double] = Map()
    var maxNTD = 0d

    //todo: potential improvement is to assign traits to co-occurring GROUPS to maximise NN in each group.
    var loop = 0
    while (loop < config.permNTD) {

      var map: Map[Int, Double] = Map()
      var traitCounter = 0
      // first randomly select trait values for the -ve co-occurring species
      val negTraits = randomSelect(negCoOccur.map(_._1).toSet.size,tValues.toList)
      val combination = randomSelect(getSpeciesSet(paMatrix, negCoOccur, coList).size,
tValues.toList.diff(negTraits))

      for (r <- rowIndices) {
        traitSubMatrix(r) = paMatrix(r).map(_ * combination(traitCounter))
        map += (r -> combination(traitCounter))
        traitCounter += 1
      }

      val currentNTD = mtxStats.meanAndStdDevNTD(traitSubMatrix
        .filter(p => p.sum != 0)
        .transpose
        .filter(p => p.sum != 0)
        .transpose)

      if (currentNTD._1 > maxNTD) {
        maxNTD = currentNTD._1
        maximal = map
```

```
    }
    loop += 1
  }
  //todo: I should maybe pick the random traits first and then determine NN with the remaining ones.
  assignTraits(paMatrix, getRemainders(tValues, maximal, paMatrix.length) ++ maximal)
 }

 def assignTraits(paMatrix: Array[Array[Int]], traitMap: Map[Int, Double]): Array[Array[Double]] = {
  val tMatrix = Array.fill(paMatrix.length, paMatrix(0).length)(0d)
  for (i <- 0 to paMatrix.length - 1) tMatrix(i) = paMatrix(i).map(_ * traitMap(i))
  tMatrix
 }

 def getRemainders(allTraits: Array[Double], traitMap: Map[Int, Double], paSize: Int): Map[Int, Double] = {
  val traits = traitMap.map(_._2).toArray
  val species = traitMap.map(_._1).toArray
  val unusedTraits = allTraits.diff(traits)
  val traitlessSpecies = (0 to paSize - 1).toArray.diff(species)
  if (unusedTraits.size != traitlessSpecies.size) {
   sys.error("Oh Shit!!!\n" +
    "all traits: " + allTraits.sorted.mkString(", ") + "\n" +
    "unused traits: " + unusedTraits.toArray.sorted.mkString(", ") + "\n" +
    "species: " + traitMap.map(_._1).toArray.sorted.mkString(", ") + "\n" +
    "traitless species: " + traitlessSpecies.toArray.sorted.mkString(", ") + "\n")
  } else
   Map(shuffle(traitlessSpecies.toArray).zip(shuffle(unusedTraits.toArray)).toArray: _*)
 }

 def getSpeciesSet(paMatrix: Array[Array[Int]], negMap: Map[Int, IndexedSeq[(Int, Int, (Double, Double,
Double))]], posList: List[Set[Int]]): Set[Int] = {
  (negMap.isEmpty, posList.isEmpty) match {
   case (true, _) => (0 to paMatrix.length - 1).toSet
   case (false, _) => (0 to paMatrix.length - 1).toSet -- negMap.map(_._1).toSet
   case (_, _) => sys.error("WE HAVE A PROBLEM HUSTON!! - There is no set of species to assign traits to!")
  }
 }

 def removeAt[A](n: Int, xs: List[A]): (List[A], A) = {
  val (heads, tails) = xs.splitAt(n)
  (heads ::: tails.tail, tails.head)
 }

 def randomSelect[A](n: Int, xs: List[A]): List[A] = (n, xs) match {
  case (_, Nil) => Nil
  case (0, _) => Nil
  case (_, _) => {
   val x = removeAt(new Random().nextInt(xs.size), xs)
   x._2 :: randomSelect(n - 1, x._1)
  }
 }

 def groupByPositiveCoOccurrence(valToGroupBy: Int, pairwisePValues: IndexedSeq[(Int, Int, (Double, Double,
Double))]): Set[Int] = {
```

```
    val av = pairwisePValues
      .filter(p => p._1 == valToGroupBy & p._1 != p._2 & p._3._2 + p._3._3 <= 0.05)
      .map(f => Set(f._1, f._2))
    if (av.isEmpty) {
      Set[Int]()
    } else
      av.reduceLeft(_ ++ _)
  }

  def groupByNegativeCoOccurrence(valToGroupBy: Int, pairwisePValues: IndexedSeq[(Int, Int, (Double, Double,
Double))]): Set[Int] = {
    val av = pairwisePValues
      .filter(p => p._1 == valToGroupBy & p._1 != p._2 & p._3._2 + p._3._1 <= 0.05)
      .map(f => Set(f._1, f._2))
    if (av.isEmpty) {
      Set[Int]()
    } else
      av.reduceLeft(_ ++ _)
  }

  def removeSubsetsOf(lSet: List[Set[Int]]): List[Set[Int]] = {
    val srtList = lSet.sortBy(_.size)
    var shortList: List[Set[Int]] = Nil

    for (i <- 0 to srtList.length - 1) {
      var flag = false
      var c = i + 1
      while (c < srtList.length & flag == false) {
        if (srtList(i).subsetOf(srtList(c)))
          flag = true
        c += 1
      }
      if (!flag) shortList = srtList(i) :: shortList
    }
    shortList
  }

  def getAllPairwiseVeech(matrix: Array[Array[Int]]): immutable.IndexedSeq[(Int, Int, (Double, Double, Double))] =
{
    for (x <- 0 to matrix.length - 1; y <- 1 to matrix.length - 1)
    yield (x, y, mtxStats.veech(matrix(x), matrix(y)))
  }

  def C(n: Int, r: Int) = factorial(n) / (factorial(r) * factorial(n - r))

  def factorial(n: BigInt): BigInt = {
    @tailrec def factorialAcc(acc: BigInt, n: BigInt): BigInt = {
      if (n <= 1) acc
      else factorialAcc(n * acc, n - 1)
    }
    factorialAcc(1, n)
  }
```

```
  def shuffle[T](array: Array[T]) = {
    val buf = new ArrayBuffer[T] ++= array

    def swap(i1: Int, i2: Int) {
      val tmp = buf(i1)
      buf(i1) = buf(i2)
      buf(i2) = tmp
    }

    for (n <- buf.length to 2 by -1) {
      val k = Random.nextInt(n)
      swap(n - 1, k)
    }
    buf
  }

}
```

E.2 Scala scripts

E.2.1 CoOccurTypeIIPropLevel.scala

```
import io.Source
 import java.io.{FileWriter, BufferedWriter, File}

 /*
 *
 * This script parses the Type II sensitivity data for the co-occurrence data and summerises it for R.
 * The final product is COTypeIISensitivitySummary.csv which is used to produce
 * the Type II Plot x Prop x Percent Panels plot.
 *
 * */

 val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/CO"

 def getFiles(path: String): Iterator[File] = new File(path)
   .listFiles.filter(f => f.isFile &
   f.getName.endsWith(".log"))
   .toIterator

 def oneTailed(lt: Double) = if (lt >= 0.95) 1 else 0
 def twoTailed(lt: Double, gt: Double) = if (lt >= 0.975 | gt >= 0.975) 1 else 0
 def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

 var dimMap, plMap, spMap, propMap, nMap = Map[(Int, Int, Int), (Int, Int)]()
 val propRange = List(0, 20, 40, 60, 80, 100)
 val spRange = List(5, 10, 15, 20, 25, 30, 35)
 val plotRange = List(5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 200)

 for (file <- getFiles(path)) {
```

```
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines; if (line split ("\t")).length == 13) {
   if (hFlag) {
    hMap = buildMap(line)
    hFlag = false
   } else {
    val currentLine = line split ("\t")
    val n = currentLine(hMap("N")).toInt
    val plots = currentLine(hMap("Plots")).toInt
    val species = currentLine(hMap("Species")).toInt
    val proportion = ((n.toDouble / species.toDouble) * 100).toInt
    val dLine = currentLine.map(s => s match {
     case """"""" => Double.PositiveInfinity
     case """?""" => Double.PositiveInfinity
     case """∞""" => Double.PositiveInfinity
     case """-""" => Double.NegativeInfinity
     case """-?""" => Double.NegativeInfinity
     case """-∞""" => Double.NegativeInfinity
     case _ => s.toDouble
    })

    "id\tcBefore\tobsC\texpC\texpCStd\tlt\teq\tgt\tses\tSpecies\tPlots\tDim\tN"


    if (propRange.contains(proportion) & spRange.contains(species)) {

     if (!propMap.contains(plots, proportion, species)) {
      propMap += (plots, proportion, species) -> {
       (1,
        //          twoTailed(dLine(hMap("lt")), dLine(hMap("gt")))
        oneTailed(dLine(hMap("lt")))
        )
      }
     } else {
      propMap += (plots, proportion, species) -> {
       (propMap((plots, proportion, species))._1 + 1,
        propMap((plots, proportion, species))._2 + oneTailed(dLine(hMap("lt")))
        )
      }
     }
    }
   }
  }
}

 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
2/Summarized Data/COTypeIISensitivitySummary.tsv"))
```

```
f.write("Plots\tProp\tSpecies\tCount\tval" + "\n")

println("\nPlots\tProp\tSpecies\tCount\tval")
for (key <- propMap.keySet.toList.sorted) {
  println(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + propMap(key)._1 + "\t" +
propMap(key)._2.toDouble / propMap(key)._1
  )

  f.write(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + propMap(key)._1 + "\t" +
propMap(key)._2.toDouble / propMap(key)._1 + "\n"
  )

  f.flush()
}

f.flush()
f.close()

def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

E.2.2 NTDtypeIISensParser.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}

/*
*
* This script parses the Type II sensitivity data and summerises it for R.
* The final product is TypeIISensitivitySummary.csv which is used to produce
* the Type II Plot x Prop x Percent Panels plot.
*
* */

val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/NN"

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

def oneTailed(lt: Double, gt: Double) = if (lt >= 0.95) 1 else 0
def checkSES(ses: Double) = if (math.abs(ses) > 1.96d) 1 else 0

var dimMap, plMap, spMap, propMap, nMap = Map[(Int, Int, Int), (Int, Int, Int)]()
val propRange = List(0, 20, 40, 60, 80, 100)
val spRange = List(5, 10, 15, 20, 25, 30, 35)
val plotRange = List(5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 200)

for (file <- getFiles(path)) {
```

```
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines) {
   if (hFlag) {
    hMap = buildMap(line)
    hFlag = false
   } else {
    val currentLine = line split ("\t")
    val n = currentLine(hMap("N")).toInt
    val plots = currentLine(hMap("plots")).toInt
    val species = currentLine(hMap("species")).toInt
    val dim = plots * species
    val proportion = ((n.toDouble / species.toDouble) * 100).toInt
    val dLine = currentLine.map(s => s match {
     case """"""" => Double.PositiveInfinity
     case """?""" => Double.PositiveInfinity
     case """∞""" => Double.PositiveInfinity
     case """-""" => Double.NegativeInfinity
     case """-?""" => Double.NegativeInfinity
     case """-∞""" => Double.NegativeInfinity
     case _ => s.toDouble
    })

    "id\tobservedC\tobsMean\tobsSDNN\t" +
     "expAITS\texpAITSStDev\texpAITSpLT\texpAITSpE\texpAITSpGT\texpAITSses\t" +
     "expAITSSDNNpLT\texpAITSSDNNpE\texpAITSSDNNpGT\texpAITSSDNNses\t" +
     "expAWTS\texpAWTSStDev\texpAWTSpLT\texpAWTSpE\texpAWTSpGT\texpAWTSses\t" +
     "expAWTSSDNNpLT\texpAWTSSDNNpE\texpAWTSSDNNpGT\texpAWTSSDNNses\t" +
     "species\tplots\tdim\tN"

    if (propRange.contains(proportion) & spRange.contains(species)) {

     if (!propMap.contains(plots, proportion, species)) {
      propMap += (plots, proportion, species) -> {
       (1,
        oneTailed(dLine(hMap("expAITSpLT")), dLine(hMap("expAITSpGT"))),
        oneTailed(dLine(hMap("expAWTSpLT")), dLine(hMap("expAWTSpGT")))
        )
      }
     } else {
      propMap += (plots, proportion, species) -> {
       (propMap((plots, proportion, species))._1 + 1,
        propMap((plots, proportion, species))._2 + oneTailed(dLine(hMap("expAITSpLT")),
dLine(hMap("expAITSpGT"))),
        propMap((plots, proportion, species))._3 + oneTailed(dLine(hMap("expAWTSpLT")),
dLine(hMap("expAWTSpGT")))
        )
      }
```

```
      }
     }
    }
   }
  }
```

```
val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 2/Summarized Data/TraitTypeIISensitivitySummary.tsv"))
//val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Data/Summarized Data/TypeIISensitivitySummary.csv"))
f.write("Plots\tSpecies\tCount\ttype\tval" + "\n")

//println("\nPlots\tProp\tSpecies\tCount\tAITSNTD\tAWTSNTD\tAITSSDNN\tAWTSSDNN")
println("\nPlots\tSpecies\tCount\ttype\tval")
for (key <- propMap.keySet.toList.sorted) {
  println(
    key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AITSNTD\t" + propMap(key)._2.toDouble / propMap(key)._1 + "\n" +
    key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AWTSNTD\t" + propMap(key)._3.toDouble / propMap(key)._1
  )

  f.write(
    key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AITSNTD\t" + propMap(key)._2.toDouble / propMap(key)._1 + "\n" +
    key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AWTSNTD\t" + propMap(key)._3.toDouble / propMap(key)._1 + "\n"
  )

  f.flush()
}

f.flush()
f.close()

def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

E.2.3 SDNNtypeIISensParser.scala

```
import io.Source
import java.io.{FileWriter, BufferedWriter, File}

/*
*
* This script parses the Type II sensitivity data and summerises it for R.
* The final product is TypeIISensitivitySummary.csv which is used to produce
* the Type II Plot x Prop x Percent Panels plot.
*
* */

val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/Type II Sensitivity Data/SDNN"
```

```
def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

def oneTailed(lt: Double, gt: Double) = if (gt >= 0.95) 1 else 0 // This needs to gt >= 0.95 for sdnn

var dimMap, plMap, spMap, propMap, nMap = Map[(Int, Int, Int), (Int, Int, Int)]()
val propRange = List(0, 20, 40, 60, 80, 100)
val spRange = List(5, 10, 15, 20, 25, 30, 35)
val plotRange = List(5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 200)

for (file <- getFiles(path)) {
  println(file.getName)

  val lines = Source.fromFile(file).getLines() //.drop(1) //removes the header from the file

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines) {
    if (hFlag) {
      hMap = buildMap(line)
      hFlag = false
    } else {
      val currentLine = line split ("\t")
      val n = currentLine(hMap("N")).toInt
      val plots = currentLine(hMap("plots")).toInt
      val species = currentLine(hMap("species")).toInt
      val dim = plots * species
      val proportion = ((n.toDouble / species.toDouble) * 100).toInt
      val dLine = currentLine.map(s => s match {
        case """"""" => Double.PositiveInfinity
        case """?""" => Double.PositiveInfinity
        case """∞""" => Double.PositiveInfinity
        case """-"""" => Double.NegativeInfinity
        case """-?""" => Double.NegativeInfinity
        case """-∞""" => Double.NegativeInfinity
        case _ => s.toDouble
      })

      "id\tobservedC\tobsSDNN\t" +

"expAITSSDNN\texpAITSSDNNStDev\texpAITSSDNNpLT\texpAITSSDNNpE\texpAITSSDNNpGT\texpAITSSDNNses
\t" +

"expAWTSSDNN\texpAWTSSDNNStDev\texpAWTSSDNNpLT\texpAWTSSDNNpE\texpAWTSSDNNpGT\texpAWT
SSDNNses\t" +
        "species\tplots\tdim\tN"

      if (propRange.contains(proportion) & spRange.contains(species)) {
```

```
      if (!propMap.contains(plots, proportion, species)) {
        propMap += (plots, proportion, species) -> {
          (1,
            oneTailed(dLine(hMap("expAITSSDNNpLT")), dLine(hMap("expAITSSDNNpGT"))),
            oneTailed(dLine(hMap("expAWTSSDNNpLT")), dLine(hMap("expAWTSSDNNpGT")))
            )
        }
      } else {
        propMap += (plots, proportion, species) -> {
          (propMap((plots, proportion, species))._1 + 1,
            propMap((plots, proportion, species))._2 + oneTailed(dLine(hMap("expAITSSDNNpLT")),
dLine(hMap("expAITSSDNNpGT"))),
            propMap((plots, proportion, species))._3 + oneTailed(dLine(hMap("expAWTSSDNNpLT")),
dLine(hMap("expAWTSSDNNpGT")))
            )
        }
      }
    }
  }
 }

 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
2/Summarized Data/TraitTypeIISensitivitySummary.tsv", true))
 //val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Data/Summarized
Data/TypeIISensitivitySummary.csv"))
 //f.write("Plots\tSpecies\tCount\ttype\tval" + "\n")

 //println("\nPlots\tProp\tSpecies\tCount\tAITSNTD\tAWTSNTD\tAITSSDNN\tAWTSSDNN")
 println("\nPlots\tSpecies\tCount\ttype\tval")
 for (key <- propMap.keySet.toList.sorted) {
  println(
    key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AITSSDNN\t" + propMap(key)._2.toDouble /
propMap(key)._1 + "\n" +
     key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AWTSSDNN\t" +
propMap(key)._3.toDouble / propMap(key)._1
  )

  f.write(
    key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AITSSDNN\t" + propMap(key)._2.toDouble /
propMap(key)._1 + "\n" +
     key._1 + "\t" + key._3 + "\t" + propMap(key)._1 + "\t" + key._2 + "-AWTSSDNN\t" +
propMap(key)._3.toDouble / propMap(key)._1 + "\n"
  )

  f.flush()
 }

 f.flush()
 f.close()

 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

Appendix F Chapter 3.0 R code

F.1.1 'CoOcc Type II x Prop Contour.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 2/Summarized
 Data/COTypeIISensitivitySummary.tsv", header = TRUE, sep = "\t")

str(d)

quartz(width = 3.0, height = 7.5)

rgb.palette <- colorRampPalette(c("red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.75, lines=1.0)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$Prop
 , levels = c("0", "20", "40", "60", "80", "100")
 , labels = c("0%", "20%", "40%", "60%", "80%", "100%")
  ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 6)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , labels = c("5","10","15","20","25","30","35")
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "")
        , cex = 0.7))
           , between = list(x = 0.25, y = 0.25))
```

F.1.2 'Trait Type II x Prop Contour.R'

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 2/Summarized
 Data/TraitTypeIISensitivitySummary.tsv", header = TRUE, sep = "\t")

str(d)

quartz(width = 5.9, height = 7.5)

rgb.palette <- colorRampPalette(c("red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.6, lines=1.0)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$type
 , levels = c("0-AITSNTD","0-AITSSDNN","0-AWTSNTD","0-AWTSSDNN",
                              "20-AITSNTD","20-AITSSDNN","20-AWTSNTD","20-AWTSSDNN",
                              "40-AITSNTD","40-AITSSDNN","40-AWTSNTD","40-AWTSSDNN",
                              "60-AITSNTD","60-AITSSDNN","60-AWTSNTD","60-AWTSSDNN",
                              "80-AITSNTD","80-AITSSDNN","80-AWTSNTD","80-AWTSSDNN",
                              "100-AITSNTD","100-AITSSDNN","100-AWTSNTD","100-AWTSSDNN")
 , labels = c("0%: AITS & NN","0%: AITS & SDNN","0%: AWTS & NN","0%: AWTS & SDNN",
           "20%: AITS & NN","20%: AITS & SDNN","20%: AWTS & NN","20%: AWTS & SDNN",
           "40%: AITS & NN","40%: AITS & SDNN","40%: AWTS & NN","40%: AWTS & SDNN",
           "60%: AITS & NN","60%: AITS & SDNN","60%: AWTS & NN","60%: AWTS & SDNN",
           "80%: AITS & NN","80%: AITS & SDNN","80%: AWTS & NN","80%: AWTS & SDNN",
           "100%: AITS & NN","100%: AITS & SDNN","100%: AWTS & NN","100%: AWTS & SDNN")
  )
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(4, 6)
 , scales = list(y = list(tck = c(0.5, 0)
          , alternating = FALSE
          , labels = c("5","10","15","20","25","30","35")
          , cex = 0.7)
          , x = list(tck = c(0.5, 0)
          , rot=90
          , alternating = FALSE
          , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "", "75", "", "150", "")
```

```
, cex = 0.65))
   , between = list(x = 0.25, y = 0.25))
```

Appendix G Chapter 4.0 supplemental figures

G.1 Limiting similarity: NN

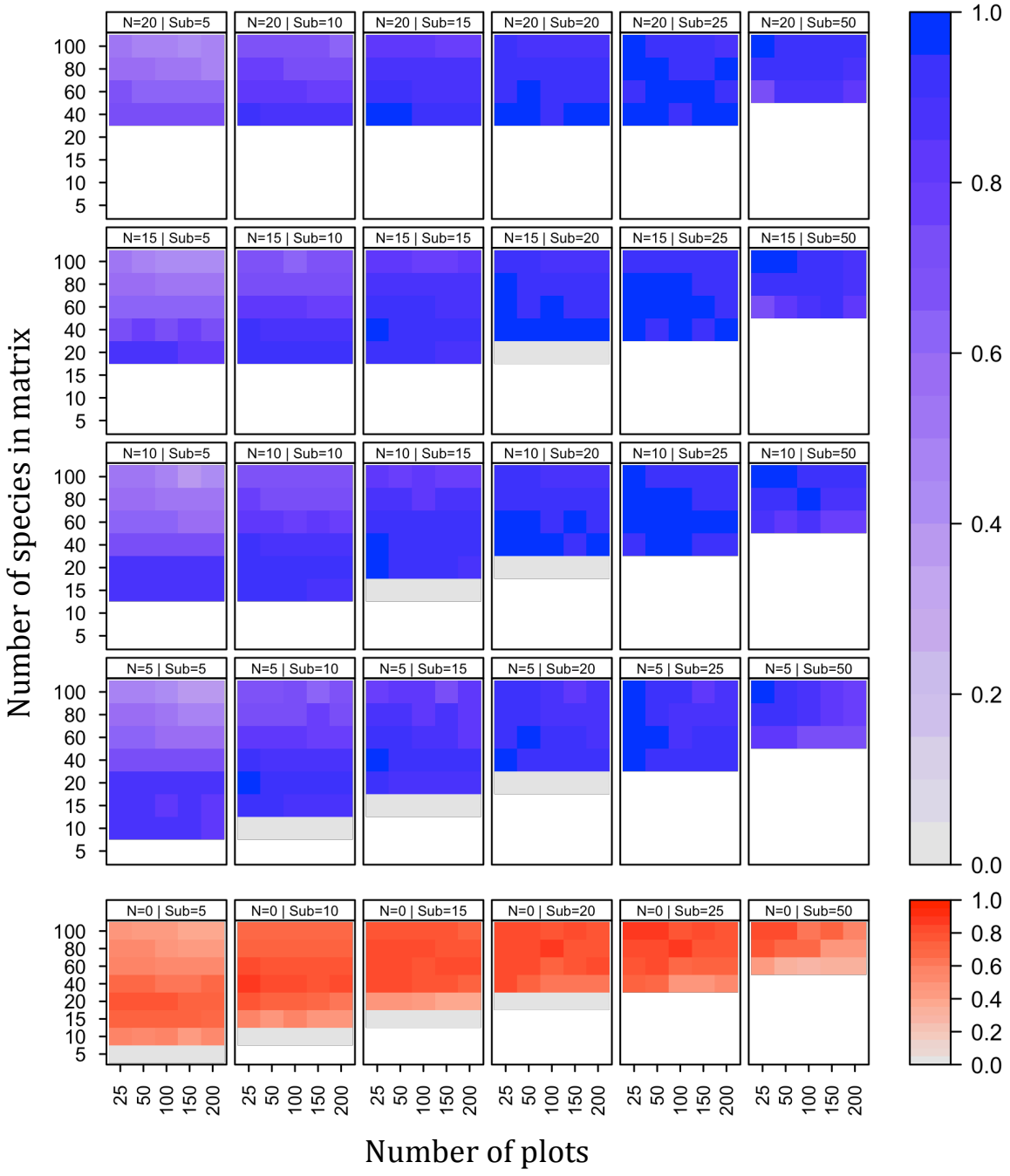

Figure G-1. The mean proportion of species by matrix dimension, subset size and number of species with signal added that indicator species analysis indicated as having strong associations with groups of species involved in significant patterns of limiting similarity. The metric used for these analyses was the nearest neighbour trait distance (NN). Blue panels represent proportions when signal had been added to the matrices. Red panels are the mean proportions of species for matrices to which no signal had been added. Values of *N* are the number of species with signal added and *Sub* are the size of the species subsets that the null model was run on. White cells in plots indicate no data collected and are associated with matrices with fewer species than required for the subset size.

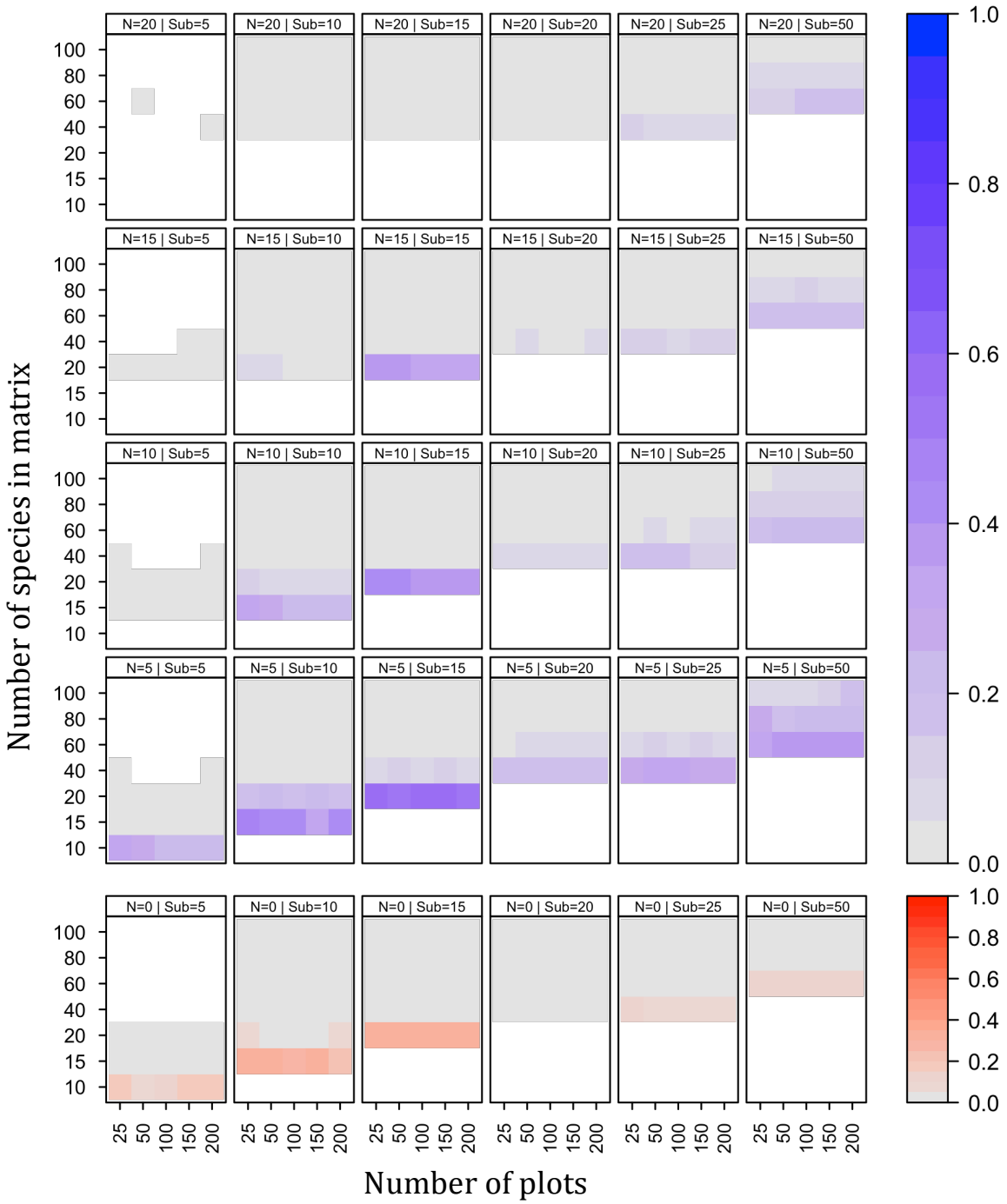
N=20 | Sub=5
N=20 | Sub=10
N=20 | Sub=15
N=20 | Sub=20
N=20 | Sub=25
N=20 | Sub=50
N=15 | Sub=5
N=15 | Sub=10
N=15 | Sub=15
N=15 | Sub=20
N=15 | Sub=25
N=15 | Sub=50
N=10 | Sub=5
N=10 | Sub=10
N=10 | Sub=15
N=10 | Sub=20
N=10 | Sub=25
N=10 | Sub=50
N=5 | Sub=5
N=5 | Sub=10
N=5 | Sub=15
N=5 | Sub=20
N=5 | Sub=25
N=5 | Sub=50
N=0 | Sub=5
N=0 | Sub=10
N=0 | Sub=15
N=0 | Sub=20
N=0 | Sub=25
N=0 | Sub=50
Number of species in matrix
100
80
60
40
20
15
10
25
50
100
150
200
Number of plots
1.0
0.8
0.6
0.4
0.2
0.0

Figure G-2. The mean proportion of species by matrix dimension, subset size and number of species with signal added that frequent pattern mining indicated as having strong associations with groups of species involved in significant patterns of limiting similarity. The metric used for these analyses was the nearest neighbour trait distance (NN). Blue panels represent proportions when signal had been added to the matrices. Red panels are the mean proportions of species for matrices to which no signal had been added. Values of *N* are the number of species with signal added and *Sub* are the size of the species subsets that the null model was run on. White cells in plots indicate no data collected and are associated with matrices that have fewer species than required for the subset size. Larger matrices with white cells are due to insufficient data for frequent pattern mining.

Appendix H Chapter 4.0 Scala Code

H.1 Application Code

H.1.1 FactorialMap

```
package ca.mikelavender.CommunityCompetition.common

 object FactorialMap {
  /**
   * This is a map of previously calculated factorial values
   */
  private var fact: Map[Int, BigInt] = Map(0 -> 1, 1 -> 1)


  /**
   * Calculate the factorial of n and stores it in the map for future (faster) look-ups.
   *
   * @param n   Int: Value we want the factorial of.
   */
  def build(n: Int) {
    lazy val factsWithPrefix: Stream[BigInt] = BigInt(0) #:: BigInt(1) #:: (factsWithPrefix.zipWithIndex.tail map {
      case (f, i) => f * i
    })

    val facts = factsWithPrefix drop 2

    var counter = 1
    facts take n foreach (f => {
      fact += (counter -> f)
      counter += 1
    })
  }

  /**
   * Returns the factorial of n.
   *
   * @param n   Int: Value we want the factorial of.
   * @return    BigInt: factorial of n
   */
  def get(n: Int) = {
    if (!fact.contains(n)) build(n)
    if (n <= 1) BigInt(1)
    else fact(n)
  }
}
```

H.1.2 CLCriterion.scala

```
package ca.mikelavender.CommunityCompetition.cooccurrence.pairwise.clcriterion

 import ca.mikelavender.nullmodeller.MatrixRandomisation

 class CLCriterion {
  val randomizer = new MatrixRandomisation

  /**
   * Given a PA matrix
   * 1. Calculate all pairwise co-occurrence values and store. (Observed)
   * 2. Randomize the matrix (fixed-fixed).
   * 3. Calculate all pairwise co-occurrence values and store. (expected)
   * 4. Repeat 2 - 3 'X' times (default = 5000).
   * 5. Calculate P values for Observed versus Expected
   */

  def getAllPairwiseNullModel(pa: Array[Array[Int]], iterations: Int = 1000): Map[(Int, Int), (Double, Double,
Double)] = {

    // Get our "Observed" set for the null model
    var resultMap = getAllPairwise(pa, Map[Tuple2[Int, Int], List[Int]]())

    // Now loop through 'x' times to get the null distribution
    var loop = 0
    while (loop < iterations) {
     // this uses the default of 30,000 swaps
     resultMap = getAllPairwise(randomizer.coOccFixedFixed(pa.map(_.clone())), resultMap)
     loop += 1
    }
    calculatePValues(resultMap, iterations)

  }

  // get all pairwise co-occurrence values and stores them in the map. The first value in is the Observed value.
  private def getAllPairwise(paMatrix: Array[Array[Int]], map: Map[Tuple2[Int, Int], List[Int]]): Map[(Int, Int),
List[Int]] = {
    var newMap = map
    for (i <- 0 to paMatrix.length - 1; j <- i + 1 to paMatrix.length - 1) {
     if (map.isEmpty) {
       newMap += (i, j) -> {
        countCoOccurrences(paMatrix(i), paMatrix(j)) :: List()
       }
     } else {
       newMap += (i, j) -> {
        countCoOccurrences(paMatrix(i), paMatrix(j)) :: map((i, j))
       }
     }
    }
    newMap
  }
```

```
  // Returns a map of species pairs and their p-values (sp_i, sp_j) => (lt, eq, gt)
  private def calculatePValues(resultMap: Map[(Int, Int), List[Int]], iterations: Int): Map[(Int, Int), (Double, Double,
Double)] = {
    resultMap.map(f => f._1 -> {
      val l = f._2.reverse
      (l.tail.count(p => p > l.head).toDouble / iterations,
        l.tail.count(p => p == l.head).toDouble / iterations,
        l.tail.count(p => p < l.head).toDouble / iterations)
    }
    )
  }

  // Returns co-occurrence counts between species pairs
  private def countCoOccurrences(a1: Array[Int], a2: Array[Int]): Int = (a1, a2).zipped.map(_ * _).sum

}
```

H.1.3 VeechPairwise.scala

```
package ca.mikelavender.CommunityCompetition.cooccurrence.pairwise.veech

 import java.text.DecimalFormat
 import ca.mikelavender.CommunityCompetition.common.FactorialMap


 class VeechPairwise {

  val factorialMap = FactorialMap

  def getSetOfAllPairwise(mtxObj: Array[Array[Int]]): Map[(Int, Int), (Double, Double, Double)] = {
    val numSpecies = mtxObj.length
    var resultMap = Map[(Int, Int), (Double, Double, Double)]()
    for (x <- 0 to numSpecies - 1; y <- x + 1 to numSpecies - 1)
      resultMap += ((x, y) -> veech(mtxObj(x), mtxObj(y)))
    resultMap
  }

  private def getAllPairwise(mtxObj: Array[Array[Int]]): Array[Array[String]] = {
    val numSpecies = mtxObj.length
    val resultArray = Array.fill[String](numSpecies, numSpecies)("#")
    for (x <- 0 to numSpecies - 1; y <- x + 1 to numSpecies - 1)
      resultArray(x)(y) = isSignificant(veech(mtxObj(x), mtxObj(y)))
    resultArray
  }

  /**
   * Calculates pairwise probabilistic species co-occurrence using the method of Veech (2012).
   * For details see:
   * Veech, J. A. (2012). A probabilistic model for analysing species co-occurrence. (P. Peres-Neto, Ed.)
   * Global Ecology and Biogeography, n/a–n/a. doi:10.1111/j.1466-8238.2012.00789.x
   *
```

```
   * @param s1  One row of data from a presence-absence matrix. Each row represents a single species
   * @param s2  The species that s1 is being compared to (another row of data)
   * @return    A tuple consisting of the exact probabilities. Values are <, =, & > the number of co-occurrences.
   */
  private def veech(s1: Array[Int], s2: Array[Int]): (Double, Double, Double) = {

    val sizeHint = s1.length
    val l: Array[Int] = new Array(sizeHint)
    var c = 0
    while (c < sizeHint) {
      l(c) = s1(c) * s2(c)
      c += 1
    }

    val N = s1.length //number of sites (plots)

    val N1 = s1.count(p => p == 1) // number of species 1
    val N2 = s2.count(p => p == 1) // number of species 2
    val co = l.count(p => p == 1) // number of co-occurrences of sp1 & sp2

    def exact(i: Int): (Double, Double, Double) = {
      ( {
        for (j <- 0 to i - 1)
        yield
          (nChooseR(N, j) * nChooseR(N - j, N2 - j) * nChooseR(N - N2, N1 - j)).toDouble / (nChooseR(N, N2) *
nChooseR(N, N1)).toDouble
      }
        .sum, {
        (nChooseR(N, i) * nChooseR(N - i, N2 - i) * nChooseR(N - N2, N1 - i)).toDouble / (nChooseR(N, N2) *
nChooseR(N, N1)).toDouble
      }, {
        for (j <- i + 1 to N)
        yield
          (nChooseR(N, j) * nChooseR(N - j, N2 - j) * nChooseR(N - N2, N1 - j)).toDouble / (nChooseR(N, N2) *
nChooseR(N, N1)).toDouble
      }
        .sum
        )
    }

    exact(co)
  }

  /**
   * Retrieves factorial for n i.e. n! from the FactorialMap object.
   *
   * @param n Integer value that we are calculating the factorial of.
   * @return  BigInt
   */
  private def factorial(n: Int): BigInt = factorialMap.get(n)

  /**
   * Implementation of n choose r.
```

```
 *
 * @param n number of choices
 * @param r number of elements we want to choose
 * @return  BigInt
 */
private def nChooseR(n: Int, r: Int): BigInt = factorial(n) / (factorial(r) * factorial(n - r))

/**
 * Tests the results for significance in either tail.
 *
 * @param pairwiseTests Tuple3[Double, Double, Double]: Result returned from method veech.
 * @return              String. possible returned values are: 'neg', 'pos', '   '.
 */
private def isSignificant(pairwiseTests: Tuple3[Double, Double, Double]) = {
  val formatter = new DecimalFormat("#0.000")
  val lt = pairwiseTests._2 + pairwiseTests._1
  val gt = pairwiseTests._2 + pairwiseTests._3
  if (lt < 0.05)
    "-" //+ formatter.format(lt)
  else if (gt < 0.05)
    "+" //+ formatter.format(gt)
  else
    "  "
}

//  private var fact: Map[Int, BigInt] = Map(0 -> 1, 1 -> 1)
//
//
// /**
//   * Calculate the factorial of n and stores it in the map for future (faster) look-ups.
//   *
//   * @param n   Int: Value we want the factorial of.
//   */
//  def build(n: Int) {
//    lazy val factsWithPrefix: Stream[BigInt] = BigInt(0) #:: BigInt(1) #:: (factsWithPrefix.zipWithIndex.tail map {
//      case (f, i) => f * i
//    })
//
//    val facts = factsWithPrefix drop 2
//
//    var counter = 1
//    facts take n foreach (f => {
//      fact += (counter -> f)
//      counter += 1
//    })
//  }
//
// /**
//   * Returns the factorial of n.
//   *
//   * @param n   Int: Value we want the factorial of.
//   * @return    BigInt: factorial of n
//   */
```

```
  // def getFact(n: Int) = {
  //   if (!fact.contains(n)) build(n)
  //   if (n <= 1) BigInt(1)
  //   else fact(n)
  // }
}
```

H.1.4 Config.scala

```
package ca.mikelavender.CommunityCompetition.frequentitemsets.fpgrowth

object Config {
  var minSupp = 1
  var transCount = 10000

  // values for hypergeo calculaton
  var hyperMap = Map[Int, Int]().withDefaultValue(0)

}
```

H.1.5 FPNode.scala

```
package ca.mikelavender.CommunityCompetition.frequentitemsets.fpgrowth

import collection.mutable.HashMap


/**
 * Created by IntelliJ IDEA.
 * User: phi
 * Date: 9/10/11
 * Time: 12:53 PM
 * To change this template use File | Settings | File Templates.
 */
class FPNode(val id: Int, var parent: FPNode) {
  var frequency = 0
  var next: FPNode = null
  var children = new HashMap[Int, FPNode]

  override def toString: String = "id: %s freq %d" format(id, frequency)


  def strVal(mappings: Array[HeaderItem], spaces: String = ""): String = {
    var accum = ""
    if (id != (-1)) {
      accum = "%s%s: %d\n" format(spaces, mappings(id).item, frequency)
    } else {
      accum = "root\n"
    }
```

```
    children.foreach((x) => {
      accum += x._2.strVal(mappings, spaces + "    ")
    })
    return accum

  }

  def add(pattern: List[Tuple2[Int, Int]], index: Int, tree: FPTree): Boolean = {
    if (pattern.size == index + 1 && this.id == pattern(index)._1) {
      frequency += pattern(index)._2
    } else {
      if (!children.contains(pattern(index + 1)._1)) {
        val node = new FPNode(pattern(index + 1)._1, this)
        children.put(pattern(index + 1)._1, node)
        tree.insertHeader(node)
      }
      frequency += pattern(index)._2
      children(pattern(index + 1)._1).add(pattern, index + 1, tree)

    }
    return true
  }
}
```

H.1.6 FPTree.scala

```
package ca.mikelavender.CommunityCompetition.frequentitemsets.fpgrowth

import scala.collection.mutable
import scala.annotation.tailrec
import scala.collection.mutable.ListBuffer
import ca.mikelavender.CommunityCompetition.common.FactorialMap


case class HeaderItem(id: Int, item: String, frequency: Int) {
  var node: FPNode = null
}

class FPRunner(arr: List[List[String]], relSupport: Double = 0.74) {

  private val tup = arr.map((li) => li.map((s) => (s, 1)))
  private val tree = new FPTree()

  Config.transCount = arr.length
  Config.minSupp = (arr.length.toDouble * relSupport).toInt
  tree.init(tup, relSupport)

  def process: List[(List[String], Int)] = tree.fpGrowth(relSupport = this.relSupport)
}

class FPTree {
```

```
val root = new FPNode(-1, null)
var headerTable: Array[HeaderItem] = null
val mappingTable = new mutable.HashMap[String, Int]
var baseTree: FPTree = null
val permute = new Permutation[(String, Int)]
var support = 1

def init(dataSource: List[List[(String, Int)]], relSupport: Double, bt: FPTree = null) {

 baseTree = bt
 support = (dataSource.length.toDouble * relSupport).toInt
 var ht = new mutable.ListBuffer[HeaderItem]
 val header = new mutable.HashMap[String, Int]

 for (trans <- dataSource) {
  for (item <- trans) {
   if (!header.contains(item._1)) {
    header.put(item._1, item._2)
   } else {
    header(item._1) += item._2
   }
  }
 }

 var count = 0
 header.toList
  .filter(x => x._2 >= support)
  .sortWith(_._2 < _._2)
  .foreach(h => {
  ht += new HeaderItem(count, h._1, h._2)
  mappingTable.put(h _1, count)
  count += 1
 })

 headerTable = ht.toArray

 for (trans <- dataSource) {
  val transaction = trans
   .filter(items => mappingTable.contains(items._1))
   .map(item => (mappingTable(item._1), item._2))
   .sortWith(_._1 > _._1)

  if (transaction.size > 0) {
   if (!this.root.children.contains(transaction(0)._1)) {
    this.root.children.put(transaction(0)._1, new FPNode(transaction(0)._1, this.root))
    insertHeader(root.children(transaction(0)._1))
   }
   root.children(transaction(0)._1).add(transaction, 0, this)
  }
 }
}

def insertHeader(fpNode: FPNode) = {
```

```
  var curr: FPNode = this.headerTable(fpNode.id).node
  if (curr == null) {
    this.headerTable(fpNode.id).node = fpNode
  } else {
    while (curr.next != null) {
      curr = curr.next
    }
    curr.next = fpNode
  }
}

def conditionalTreeDatasource(link: FPNode): List[List[(String, Int)]] = {
  var currLink = link
  var patterns = List[List[(String, Int)]]()
  while (currLink != null) {

    val nodeSupport = currLink.frequency
    var currNode = currLink.parent
    var pattern = List[(String, Int)]()
    while (currNode != root) {
      pattern = (headerTable(currNode.id).item, nodeSupport) :: pattern
      currNode = currNode.parent
    }
    if (pattern.size > 0) {
      patterns = pattern :: patterns
    }
    currLink = currLink.next
  }
  patterns
}

@tailrec
private def isSingleChain(curr: FPNode = root): Boolean = {
  curr.children.size match {
    case 0 => true
    case 1 => isSingleChain(curr.children.values.head)
    case _ => false
  }
}

def findFrequency(p: List[(String, Int)]): (List[String], Int) = {
  var result = List[String]()
  val length = p.length
  var newP = p
  var min = if (p.isEmpty) 0 else Int.MaxValue
  var i = 0
  while (i < length) {
    result = newP.head._1 :: result
    if (newP.head._2 < min) min = newP.head._2
    newP = newP.tail
    i += 1
  }
  (result.reverse, min)
```

```
  }

  def fpGrowth(pattern: List[(String, Int)] = List[(String, Int)](), relSupport: Double): List[(List[String], Int)] = {

    if (isSingleChain()) {
      var optional = List[(String, Int)]()
      if (root.children.size > 0) {
        var curr = root.children.head
        // This walks the tree (single path) from top to bottom building the full path as it goes.
        while (curr._2.children.size > 0) {
          optional = (headerTable(curr._2.id).item, curr._2.frequency) :: optional
          curr = curr._2.children.head

        }
        //this tacks on the last node because it gets skipped by the while loop (children.size = 0)
        optional = (headerTable(curr._2.id).item, curr._2.frequency) :: optional
      }
      val l = permute.permuteAll(optional, pattern.length)

      l.flatten.map(x => findFrequency(x.toList ::: pattern)).toList //.toList.sortBy(_._2).foreach(println)

    } else {
      val listRet = new ListBuffer[Tuple2[List[String], Int]]
      for (x <- 0 until headerTable.size) {
        val i = headerTable.size - 1 - x
        // I don't get this next line. Why filter by less than minimum support?
        if (support <= headerTable(i).frequency) {
          var patternBase: List[(String, Int)] = pattern
          patternBase = (headerTable(i).item, headerTable(i).frequency) :: patternBase

          val condTree = new FPTree
          condTree.init(this.conditionalTreeDatasource(headerTable(i).node),
(conditionalTreeDatasource(headerTable(i).node).length * relSupport).toInt)

          val v = findFrequency(patternBase) :: condTree.fpGrowth(patternBase, relSupport)
//          v.filter(p => p._2 > Config.hyperMap(p._1.length) && p._1.length > 1).foreach(println)
          listRet.appendAll(v.filter(p => p._2 > Config.hyperMap(p._1.length) ))
        }
      }
      listRet.toList.sortBy(x => (x._1.length, x._2))
    }
  }

  def hypergeoP(N: Int, K: Int, n: Int, k: Int): BigDecimal = {
    if (k == n + 1) 0
    else choose(K, k) * choose(N - K, n - k) / choose(N, n) + hypergeoP(N, K, n, k + 1)
  }

  def hypergeoD(N: Int, K: Int, n: Int, k: Int): BigDecimal = {
    if (k == n + 1) 0
    else choose(K, k) * choose(N - K, n - k) / choose(N, n)
  }
```

```
  val factorialMap = FactorialMap

  def fact(n: Int): BigInt = factorialMap.get(n)

  def choose(n: Int, k: Int): BigDecimal = BigDecimal(fact(n)) / BigDecimal(fact(k)) / BigDecimal(fact(n - k))
}
```

H.1.7 Permutation.scala

```
package ca.mikelavender.CommunityCompetition.frequentitemsets.fpgrowth

class Permutation[T] {

  def permuteAll(optional: List[(String, Int)], prefixPathLen: Int): Iterator[Iterator[Set[(String, Int)]]] = {
    val it = (1 to optional.length).toIterator
    it.map(i => optional.toSet.filter(x => x._2 > Config.hyperMap(prefixPathLen + i)).subsets(i))
  }


}
```

H.1.8 IndVal.scala

```
package ca.mikelavender.CommunityCompetition.indicatorspecies

import scala.Array
import scala.collection.mutable.ArrayBuffer
import scala.util.Random
import scala.math._
import scala.Tuple2
import java.text.DecimalFormat

class IndVal {

  private val formatter = new DecimalFormat("#0.000")

  // private var hPrinted = false
// private val out = Logger

  def process(recordSet: List[Tuple2[String, Array[Int]]]): Map[Int, (Double, Double, Double)] = {
    var workingSet = recordSet

    // Comment out one of the lines below to switch between methods: IntSet is basic indVal calculation, GroupSet
does species groups.
    val metrics = new IntSet
    //    val metrics = new GroupSet

    // Initialize IndVal
    metrics.init(recordSet)
```

```
    val storage = Array.fill(metrics.fullSet.size)(List[Double]())

    for (nul <- 0 to 200) {
     // Does this need to be sorted?
     //     for (sp <- metrics.fullSet.toList.sorted) {
      var cnt = 0
      for (sp <- metrics.fullSet.toList) {
       if (nul == 0) {
        // Uncomment the next line to use the non-equalized version (Table 1 of paper)
        storage(cnt) = metrics.Apa(sp) :: metrics.indVal(sp) :: storage(cnt)
        //        storage(cnt) =  metrics.Agpa(sp) :: metrics.indValgpa(sp) :: storage(cnt)
       } else {
        // Uncomment the next line to use the non-equalized version (Table 1 of paper)
        storage(cnt) = metrics.Apa(sp) :: storage(cnt)
        //        storage(sp) = metrics.Agpa(sp) :: storage(cnt)
       }
       cnt += 1
      }
      workingSet = shuffle(workingSet)
      metrics.init(workingSet)
    }

    var resultMap = Map[Int, (Double, Double, Double)]()

    for (sp <- 0 to metrics.fullSet.toList.length - 1) {
     val flipped = storage(sp).reverse
     resultMap += (sp ->(flipped.tail.count(p => p > flipped.head).toDouble / flipped.size,
      flipped.tail.count(p => p == flipped.head).toDouble / flipped.size,
      flipped.tail.count(p => p < flipped.head).toDouble / flipped.size)
      )
    }

    resultMap
  }

  def prettyPrinter(i: List[String], o: List[Tuple7[Int, Int, Int, Double, Double, Double, Double]], ss: Int, sp: Int, file:
String) = {
    val pl = o.head._2
    val n = o.head._3

    val speciesContribSignal = i.map(_.toInt).sorted
    val topNIndex = o.sortBy(_._4).reverse.take(n).map(_._1)
    val topNzScore = o.sortBy(_._7).reverse.take(n).map(_._1)
    val allSigzScore = o.filter(f => abs(f._7) >= 1.93).sortWith(_._7 > _._7).map(_._1)

    val topNIndex_match = topNIndex.toSet.intersect(speciesContribSignal.toSet)
    val topNzScore_match = topNzScore.toSet.intersect(speciesContribSignal.toSet)
    val allSigzScore_match = allSigzScore.toSet.intersect(speciesContribSignal.toSet)

    val topNIndex_percent = topNIndex_match.size.toDouble / speciesContribSignal.length.toDouble
    val topNzScore_percent = topNzScore_match.size.toDouble / speciesContribSignal.length.toDouble
    val allSigzScore_percent = allSigzScore_match.size.toDouble / speciesContribSignal.length.toDouble
    val allSigzScore_length = allSigzScore.size.toDouble / speciesContribSignal.length.toDouble
```

```
    val stringToPrint =
      sp + "\t" +
        pl + "\t" +
        n + "\t" +
        ss + "\t" +
        formatter.format(topNIndex_percent) + "\t" +
        formatter.format(topNzScore_percent) + "\t" +
        formatter.format(allSigzScore_percent) + "\t" +
        formatter.format(allSigzScore_length) + "\t" +
        speciesContribSignal + "\t" +
        o.sortBy(_._4).reverse.take(n).map(a => a._1) + "\t" +
        o.sortBy(_._7).reverse.take(n).map(a => a._1) + "\t" +
        file

    //    if (out.appendFlag == false) {
    //      println(header)
    //      out.writeToFile(header)
    ////      hPrinted = true
    //    }
    println(stringToPrint)
//    out.writeToFile(stringToPrint)

  }

  def shuffle(list: List[Tuple2[String, Array[Int]]]) = {
    val buf = new ArrayBuffer[Tuple2[String, Array[Int]]]() ++= list

    def swap(i1: Int, i2: Int) {
      val tmp1 = buf(i1)
      val tmp2 = buf(i2)
      buf(i1) = tmp2.copy(_2 = tmp1._2)
      buf(i2) = tmp1.copy(_2 = tmp2._2)
    }

    for (n <- buf.length to 2 by -1) {
      val k = Random.nextInt(n)
      swap(n - 1, k)
    }
    buf.toList
  }

  def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

  def stdDev(numbers: List[Double]): Double = {

    var sum: Double = 0.0
    if (numbers.length >= 2) {
      val avg = mean(numbers)
      val factor: Double = 1.0 / (numbers.length.toDouble - 1)
      for (x: Double <- numbers) {
```

```
      sum = sum + ((x - avg) * (x - avg))
    }
    sum = sum * factor
  }
  sqrt(sum)
 }

 def zScore(numbers: List[Double]) = (numbers.head - mean(numbers.tail)) / stdDev(numbers.tail)

 //
 // def writeToFile(string: String) {
 //   val out = new FileWriter(outfile, appendFlag)
 //   appendFlag = true
 //   try {
 //     out.write(string + "\n")
 //   } finally {
 //     out.close()
 //   }
 // }

}
```

H.1.9 Metrics.scala

```
package ca.mikelavender.CommunityCompetition.indicatorspecies

import collection.mutable.HashMap


/**
 * This file is copyright (c) 2012-2013 T. Mike Lavender
 */

/**
 * This is the implementation of the PCF and PSF algorithms for indicator species analyses (Cáceres and Legendre 2009).
 * They are described here:
 * <br/><br/>
 * <p/>
 * Cáceres, M. D., and P. Legendre. 2009. Associations between species and groups of sites: indices and statistical inference. Ecology 90:3566–3574.
 * <br/><br/>
 * <p/>
 * This is an optimized version that saves the result to a file
 * or keep it into memory if no output path is provided
 * by the user to the runAlgorithm method().
 *
 * @author T. Mike Lavender
 *
 */
```

```
abstract class Metrics {

  type T

  // These are the maps that store the counts of each item with respect to group.
  val gtMap, nsMap = new HashMap[T, Double]().withDefaultValue(0d)

//  for (i <- 0 to spCount - 1) {
//    gtMap.update(i, 0d)
//    nsMap.update(i, 0d)
//  }


  // These are the counters for the number of sites (subsets) per group.
  var gtCount, nsCount, sites = 0d

  // This is the full set of species for all sites.
  var fullSet = Set[T]()


  /* A from paper */
  def Apa(spID: T) = gtMap(spID)/(gtMap(spID) + nsMap(spID))

  def Agpa(spID: T) = Bpa(spID) / ((gtMap(spID) / gtCount) + (nsMap(spID) / nsCount))

  def Bpa(spID: T) = gtMap(spID) / gtCount

  def indVal(spID: T) = math.sqrt(Apa(spID) * Bpa(spID))

  def indValgpa(spID: T) = math.sqrt(Agpa(spID) * Bpa(spID))

  def init(data: List[Tuple2[String, Array[Int]]]) = {

   for (row <- data) {
    val sig = row._1
    val itemSet = row._2

    /** First increment the site counter for the group. N-sub-p = gtCount */
    sig match {
     case "gt" => gtCount += 1d
     case _ => nsCount += 1d
     //       case "lt" => ltCount += 1d
    }

    // Increment the site counter. Each row of data (subset) is a site
    sites += 1

    /** Now for each item (species) in the itemset increment the counter for it's occurrence in the group.
      * n = total gtMap(spID) + nsMap(spID), n-sub-p = gtMap(spID) */
    countOccurrences(itemSet, sig)
   }
```

```
    // Now for each item in the fullSet call the metrics methods. This calculates the values we need

    for (item <- fullSet) indVal(item)

  }

  def countOccurrences(itemSet: Array[Int], sig: String)
}

class IntSet extends Metrics {
  type T = Int

  override def countOccurrences(itemSet: Array[Int], sig: String) {
    for (item <- itemSet) {
      sig match {
        case "gt" => gtMap(item) += 1d
        case _ => nsMap(item) += 1d
        //        case "lt" => ltMap(item) += 1d
      }
      fullSet += item
    }
  }
}

class GroupSet extends Metrics {
  type T = Set[Int]

  override def countOccurrences(itemSet: Array[Int], sig: String) {
//    for (ssSize <- 5 to 10 by 5){
      for (item: Set[Int] <- itemSet.toSet.subsets(2)) {
        sig match {
          case "gt" => gtMap(item) += 1d
          case _ => nsMap(item) += 1d
          //        case "lt" => ltMap(item) += 1d
        }
        fullSet += item
      }
    }
// }
}
```

H.1.10 ISDataGenerator.scala

```
package ca.mikelavender.CommunityCompetition.isdatagenerator

import scala.util.Random
import ca.mikelavender.nullmodeller.NullModels
import scala.math._
import java.io.FileWriter
```

```
import ca.mikelavender.CommunityCompetition.{TraitsObject, MatrixObject, Config}
import ca.mikelavender.CommunityCompetition.subsettests.BuildLSTestMatrices


object ISDataGenerator {
  def main(args: Array[String]) {
    val runner = new ISDataGenerator

    val usage =
      """
        | Usage: java -jar CompetitionTests.jar [--mtxFile <PATH>] [--traitFile <PATH>] [--nullCount num] [--subsets
num]
        |    [--mtxFile <PATH>]   => matrix file (with the full path)
        |    [--traitFile <PATH>] => trait file (with the full path)
        |    [--nullCount num]    => number of null models to use in analysis.
        |    [--subsets num]      => number of subsets/size to test (default=1000)
        |    [--cpu num]         => number of cpu's (threads) to use when running. (default=1)
        |                       the default is to run as a sequential program (one thread/core).
        |                       This setting really only applies on multi-core systems and is intended
        |                       to allow other programs to share computing power of the system.
        |    [--silent]          => suppresses output to the console window.
        |    usage               => prints this message
        |
        |    Examples: java -jar CompetitionTests.jar blah blah blah
      """.stripMargin

    if (args.contains("usage")) {
      println(usage)
      sys.exit(1)
    }
    val argList = args.toList
    type OptionMap = Map[Symbol, Any]

    def nextOption(map: OptionMap, list: List[String]): OptionMap = {
      list match {
        case Nil => map
        case "--mtxFile" :: value :: tail => nextOption(map ++ Map('mtxFile -> value), tail)
        case "--traitFile" :: value :: tail => nextOption(map ++ Map('traitFile -> value), tail)
        case "--nullCount" :: value :: tail => nextOption(map ++ Map('nullCount -> value.toInt), tail)
        case "--subsets" :: value :: tail => nextOption(map ++ Map('subsets -> value.toInt), tail)
        case "--reps" :: value :: tail => nextOption(map ++ Map('reps -> value.toInt), tail)
        case "--cpu" :: value :: tail => nextOption(map ++ Map('cpu -> value.toInt), tail)
        case "--species" :: value :: tail => nextOption(map ++ Map('species -> value.toInt), tail)
        case "--plots" :: value :: tail => nextOption(map ++ Map('plots -> value.toInt), tail)
        case "--N" :: value :: tail => nextOption(map ++ Map('N -> value.toInt), tail)
        case "--type" :: value :: tail => nextOption(map ++ Map('type -> value.toString), tail)
        case "--permNTD" :: value :: tail => nextOption(map ++ Map('permNTD -> value.toInt), tail)
        case "--co_occurrence" :: value :: tail => nextOption(map ++ Map('co_occurrence -> value.toBoolean), tail)
        case "--silent" :: value :: tail => nextOption(map ++ Map('silent -> value.toBoolean), tail)
        case option :: tail => println("Unknown option " + option)
          sys.exit(1)
      }
    }
```

```
    val options = nextOption(Map(), argList)
    println(options)

    runner.run(options)
  }
}

class ISDataGenerator {
  val config = Config
  var appendFlag = false

  def run(options: Map[Symbol, Any]) {


    //   config.mtxFile = options.getOrElse('mtxFile,
"src/test/resources/paMatrix_headers.csv").asInstanceOf[String]
    //   config.traitFile = options.getOrElse('traitFile, "src/test/resources/traitMatrix.csv").asInstanceOf[String]
    //   config.mtxFile = options.getOrElse('mtxFile, "src/main/resources/testMatrix.csv").asInstanceOf[String]
    //   config.traitFile = options.getOrElse('traitFile, "src/main/resources/testTrait.csv").asInstanceOf[String]
    config.nullCount = options.getOrElse('nullCount, 5000).asInstanceOf[Int]
    config.subsets = options.getOrElse('subsets, 10000).asInstanceOf[Int]
    config.reps = options.getOrElse('reps, 1).asInstanceOf[Int]
    config.cpu = options.getOrElse('cpu, 1).asInstanceOf[Int]
    config.silent = options.getOrElse('silent, true).asInstanceOf[Boolean]

    config.species = options.getOrElse('species, 10).asInstanceOf[Int]
    config.plots = options.getOrElse('plots, 10).asInstanceOf[Int]
    config.N = options.getOrElse('N, 0).asInstanceOf[Int]
    config.typ = options.getOrElse('type, "nn").asInstanceOf[String]
    config.permNTD = options.getOrElse('permNTD, 100000).asInstanceOf[Int]
    config.alwaysRunCo_occurrence = options.getOrElse('co_occurrence, false).asInstanceOf[Boolean]

    for (i <- 1 to config.reps) {
      appendFlag = false
      config.outfile = "sp" +
        config.species +
        "p" +
        config.plots +
        "perm" +
        config.permNTD +
        "N" +
        config.N +
        "Type-" +
        config.typ +
        "_" +
        Random.nextInt(10000).toString +
        "_SubsetsFinal.v1.log"

      time(setup())
    }

  }
```

```
  def time[A](f: => A) = {
    val s = System.nanoTime
    val ret = f
    println("time: " + (System.nanoTime - s) / 1e9 / 3600 + " hours")
    writeToFile("time: " + (System.nanoTime - s) / 1e9 / 3600 + " hours")
    ret
  }

  def setup() {

    val testMaterial = BuildLSTestMatrices

    testMaterial.setup(config.species, config.plots, config.N, config.typ, config.permNTD)

    //This block of code fakes a matrix file and trait file - that is it creates synthetic data.
    val matrixObject = new MatrixObject {
      var labels: IndexedSeq[String] = _
      var header: IndexedSeq[String] = _
      var matrix: Array[Array[Int]] = _
    }

    val traitsObject = new TraitsObject {
      var labels: IndexedSeq[String] = _
      var header: IndexedSeq[String] = _
      var matrix: Array[Array[Double]] = _
    }

    matrixObject.matrix = testMaterial.paMatrix
    traitsObject.matrix = testMaterial.tMatrix
    traitsObject.header = IndexedSeq("species", "trait")
    // End of BLOCK

    if (!config.silent) {
      println(testMaterial.maxSpecies)
      for (i <- 0 to testMaterial.paMatrix.length - 1) {
        println(i + ":\t " + testMaterial.paMatrix(i).mkString(","))
      }
    }
    writeToFile(testMaterial.maxSpecies.toString())
    for (i <- 0 to testMaterial.paMatrix.length - 1) {
      writeToFile(i + ":\t " + testMaterial.paMatrix(i).mkString(","))
    }
    //    writeToFile(testMaterial.paMatrix.deep.mkString("\n"))
    writeToFile("#######")

    // This code is if I am reading a file in.
    //       val matrixObject = ft.readPA(config.mtxFile, hasHeader = true, hasRowLabels = true, ",")
    //       val traitsObject = ft.readTrait(config.traitFile, hasHeader = true, hasRowLabels = true, ",")

    println("#######")

    if (matrixObject.matrix.length != traitsObject.matrix.length) sys.error("Number of traits and number of species
do not match.")
```

```
 subsetLoop(matrixObject, traitsObject)

}

def subsetLoop(paMatrix: MatrixObject, traitMatrix: TraitsObject) {

 val nm = new NullModels

 var subset01 = 0
 val speciesIndicesList = (0 to paMatrix.matrix.length - 1).toList

 println("Starting subset loops")

 val header = "\t\t\t\t\t" + {
  traitMatrix.header.drop(1).mkString("\t\t\t\t")
 } + "\n" +
  "ID\tSP\tPL\tDIM\tCScore\t" + {
  traitMatrix.matrix(0).map(_ => "AITS_NN\tAITS_SDNN")
 }.mkString("\t") +
  "\t" + "Sp_Group"

 //   REMOVED to remove AWTS null model
 //   val header = "\t\t\t\t\t" + {
 //     traitMatrix.header.drop(1).mkString("\t\t\t\t")
 //   } + "\n" +
 //     "ID\tSP\tPL\tDIM\tCScore\t" + {
 //     traitMatrix.matrix(0).map(_ => "AITS_NN\tAITS_SDNN\tAWTS_NN\tAWTS_SDNN")
 //   }.mkString("\t") +
 //     "\t" + "Sp_Group"

 if (!config.silent) println(header)
 writeToFile(header)



 for (subsetSize <- subsetSizes(paMatrix.matrix.length)) {
  // This next loop does all proportions "subsets" times and does a FULL null model only once - last thing.
  for (subset <- 1 to {
   if (subsetSize == paMatrix.matrix.length) 1 else config.subsets
  }) {

    val y = paMatrix.matrix(0).length

    val subsetOfSpecies = randomSelect(subsetSize, speciesIndicesList)

    val testMatrixTemp = Array.ofDim[Int](subsetSize, y)
    val testTraits = Array.ofDim[Double](subsetSize, y)

    for (i <- 0 to subsetSize - 1) {
     testMatrixTemp(i) = paMatrix.matrix(subsetOfSpecies(i))
     testTraits(i) = traitMatrix.matrix(subsetOfSpecies(i))
    }
```

```
      val testMatrix = testMatrixTemp.transpose.filter(_.sum != 0).transpose

      //      if (checkConstraints(testMatrix)) {

      //Perform null models - these return a List of values, the last one is the observed value.
      //      val coNull = nm.iSwapNullAnalysisController(testMatrix, config.nullCount, config.cpu, normalized =
false).reverse

      var tNulls = List[(List[Double], List[Double])]()

      testTraits.transpose.foreach {
       case trate => tNulls = {
        val aits = nm.nullModelAITS(testMatrix, trate, config.nullCount)
        //          val awts = nm.nullModelAWTS(testMatrix, trate, config.nullCount)
        (
         aits._1.reverse,
         aits._2.reverse //,
         //           awts._1.reverse,
         //           awts._2.reverse
         ) :: tNulls
       }
      }

      val coNull = {
       if (isItWorthCScore(tNulls)) {
        nm.iSwapNullAnalysisController(testMatrix, config.nullCount, config.cpu, normalized = false).reverse
       } else
        List()
      }


      def outString: String = {
       subset + "\t" +
        subsetSize + "\t" +
        testMatrix(0).length + "\t" +
        subsetSize * testMatrix(0).length + "\t" + {
        if (coNull.isEmpty) "NA" else isSignificant(coNull.head, coNull.tail)
       } + "\t" + (
        for (t <- tNulls) yield {
         isSignificant(t._1.head, t._1.tail) + "\t" +
          isSignificant(t._2.head, t._2.tail) //+ "\t" +
         //            isSignificant(t._3.head, t._3.tail) + "\t" +
         //            isSignificant(t._4.head, t._4.tail)
        }
        ).mkString("\t") +
        "\t" + subsetOfSpecies.sorted
      }


      if (true || isItWorthCScore(tNulls)) {
       if (!config.silent) println(outString)
```

```
        writeToFile(outString)
      }

      //      print(loop + 1 + "\t")
      //      print(x + "\t")
      //      print(testMatrix(0).length + "\t")
      //      print(x * testMatrix(0).length + "\t")
      //      print(ses(coNull.head, coNull.tail) + "\t")
      //
      //      for (t <- tNulls) {
      //
      //        print(ses(t._1.head, t._1.tail) + "\t")
      //        print(ses(t._2.head, t._2.tail) + "\t")
      //
      //        print(ses(t._3.head, t._3.tail) + "\t")
      //        print(ses(t._4.head, t._4.tail) + "\t")
      //      }
      //
      //      println(subset.sorted)

      //      }

      //    }

    }
  }

}

/*
This method returns a list of integer values that represent to proportions we are interested in.
It also filters out any values that do not meet the minimum dimension of 5 species to avoid type I
errors.
*/
// def subsetSizes(n: Int): List[Int] = Set(0.05, 0.1, 0.15, 0.2, 0.4, 0.6, 0.8, 1.0).map(f => if (f * n <= 5) 5 else (f *
n).toInt).filter(_ > 0).toList.sorted
def subsetSizes(n: Int): List[Int] = (n :: List(5, 10, 15, 20, 25, 50).filter(_ < n)).sorted


/*
Answers the question as to whether we want to waste time on co-occurrence null model. It is
only worth it if we have a significant limiting similarity test.
*/
def isItWorthCScore(ts: List[(List[Double], List[Double])]) = {
  var test = false
  for (t <- ts) {
    if (isSignificant(t._1.head, t._1.tail) != "ns" ||
      isSignificant(t._2.head, t._2.tail) != "ns" || // |
      config.alwaysRunCo_occurrence
    //      ses(t._3.head, t._3.tail) != "NA" |
    //      ses(t._4.head, t._4.tail) != "NA"
    ) {
      test = true
```

```
    }
  }
  test
  false
}

def isSignificant(obs: Double, exp: List[Double]) = {
  val lt = exp.count(_ > obs).toDouble / exp.length.toDouble
  val gt = exp.count(_ < obs).toDouble / exp.length.toDouble
  var sig = "ns"


  if (gt >= 0.975) sig = "gt"
  if (lt >= 0.975) sig = "lt"

  if (gt >= 0.995) sig = "gt>"
  if (lt >= 0.995) sig = "lt>"

  if (gt >= 0.9995) sig = "gt>>"
  if (lt >= 0.9995) sig = "lt>>"

  sig
}

def ses(obs: Double, exp: List[Double]) = {
  val result = (obs - mean(exp)) / stdDev(exp)

  if (abs(result) >= 1.96)
    result
  else "NA"
}

def checkConstraints(paMatrix: Array[Array[Int]]): Boolean = {
  //    paMatrix.length * paMatrix(0).length >= 100 &
  paMatrix(0).length >= 3
  //      paMatrix(0).length <= 20

}

def randomSelect[A](n: Int, xs: List[A]): List[A] = (n, xs) match {
  case (_, Nil) => Nil
  case (0, _) => Nil
  case (_, _) => {
    val x = removeAt(new Random().nextInt(xs.size), xs)
    x._2 :: randomSelect(n - 1, x._1)
  }
}

def removeAt[A](n: Int, xs: List[A]): (List[A], A) = {
  val (heads, tails) = xs.splitAt(n)
  (heads ::: tails.tail, tails.head)
}
```

```
  def X(start: Int = 10, end: Int) = start + Random.nextInt(end - start + 1)

  def mean(numbers: List[Double]): Double = numbers.reduceLeft(_ + _) / numbers.length

  def stdDev(numbers: List[Double]): Double = {

    var sum: Double = 0.0
    if (numbers.length >= 2) {
      val avg = mean(numbers)
      val factor: Double = 1.0 / (numbers.length.toDouble - 1)
      for (x: Double <- numbers) {
        sum = sum + (x - avg) * (x - avg)
      }
      sum = sum * factor
    }
    sqrt(sum)
  }

  def writeToFile(string: String) {
    val out = new FileWriter(config.outfile, appendFlag)
    appendFlag = true
    try {
      out.write(string + "\n")
    } finally {
      out.close()
    }
  }
}
```

H.1.11 MatrixFactory.scala

```
package ca.mikelavender.CommunityCompetition.matrixFactory

 import util.Random
 import ca.mikelavender.nullmodeller.utils

 class MatrixFactory {
  private val utils = new utils

   /*Builds a random matrix using row and column constraints.
   * Row constraints - abundance for each row sampled from log-normal distribution
   * Column constraints - proportion for each column sampled from uniform distribution
   * For methods see:
   * Ulrich, W., & Gotelli, N. J. (2010). Null model analysis of species associations using abundance data. Ecology,
91(11), 3384–3397. doi:10.1890/09-2157.1
   * Gotelli, N. J. (2000). Null Model Analysis of Species Co-Occurrence Patterns. Ecology, 81(9), 2606.
doi:10.2307/177478
   * */

   //todo: I don't need to constrain the random matrix to proportion only the null matrices which are constrained by
the original random matrix proportions.
```

```
  def buildMatrix(species: Int, plots: Int): Array[Array[Int]] = buildMatrix(species, plots, "logNormal")

  def buildMatrix(species: Int, plots: Int, speciesConstraint: String): Array[Array[Int]] = {

    def go: Array[Array[Int]] = {
      val spConstArray = getSpeciesConstraints(species, plots, speciesConstraint)
      val matrix = Array.fill(species, plots)(0)
      for (spIndex <- 0 to species - 1)
        matrix(spIndex) = buildConstrainedRow(spIndex, plots, spConstArray)
      if (constrainColumns(matrix) & utils.isSwappable(matrix, species, plots)) matrix else go
    }

    go
  }

  def buildConstrainedRow(rowIndex: Int, plots: Int, speciesConstraints: Array[Int]): Array[Int] = {
    var workingRow = Array.fill(plots)(if (Random.nextDouble() <= speciesConstraints(rowIndex).toDouble / plots) 1
else 0)

    def constrainRow {
      workingRow.sum.compare(speciesConstraints(rowIndex)) match {
        case -1 => {
          val zeros = workingRow.zipWithIndex.filter(_._1 == 0).map(_._2)
          workingRow = workingRow.updated(zeros(Random.nextInt(zeros.size)), 1)
          constrainRow
        }
        case 1 => {
          val ones = workingRow.zipWithIndex.filter(_._1 == 1).map(_._2)
          workingRow = workingRow.updated(ones(Random.nextInt(ones.size)), 0)
          constrainRow
        }
        case 0 => // Nothing to do. the row meets constraints
      }
    }
    constrainRow

    workingRow
  }


  // creates a single row for the matrix using column proportion as the constraint
  def buildRow(sites: Int, colProportions: Array[Double]) = {
    for (c <- 0 to sites - 1)
    yield {
      if (Random.nextDouble() <= colProportions(sites - 1)) 1
      else 0
    }
  }

  // Makes sure that each plot has at least one species in it.
  def constrainColumns(matrix: Array[Array[Int]]): Boolean = {
    if (matrix.
      transpose.
```

```
      map(_.sum).
      zipWithIndex.
      filter(_._1 == 0).
      map(_._2).length > 0) false
    else true
  }

  def getSpeciesConstraints(species: Int, plots: Int, speciesConstraint: String): Array[Int] = {
    speciesConstraint match {
      case "logNormal" => getArrayOfNonZeroLogNormals(species, plots)
      //todo: need to implement these in helpers class
      case "normal" => sys.error("Normal distribution not implemented yet")
      case "uniform" => sys.error("Uniform distribution not implemented yet")
      case _ => sys.error("ABORT: No valid distribution given for species abundances")
    }
  }

  private def getNextLogNormal(a: Double) = math.exp(Random.nextGaussian() / (2 * a))

  def getNextLogNormal(maxVal: Int, a: Double = 0.2): Int = {
    if (maxVal <= 0) sys.error("ABORT: Invalid maximum value for log-normal distribution given")
    var l = 0
    while (l == 0) {
      val nextInt = getNextLogNormal(a).toInt
      if (nextInt != 0 & nextInt <= maxVal) l = nextInt
    }
    l
  }

  def getArrayOfNonZeroLogNormals(species: Int, plots: Int = 0, a: Double = 0.2): Array[Int] = {
    var l: List[Int] = Nil
    while (l.size < species) {
      val nextInt = getNextLogNormal(a).toInt
      if (nextInt != 0 & plots == 0) l = nextInt :: l
      else if (plots > 0 & nextInt <= plots & nextInt != 0) l = nextInt :: l
    }
    l.toArray
  }
}
```

H.1.12 Config.scala

```
package ca.mikelavender.CommunityCompetition.speciesrecoverytests

/**
 * Created with IntelliJ IDEA.
 * User: MikeLavender
 * Date: 2014-03-19
 * Time: 5:13 PM
 */
object Config {
```

```
  var typ = ""
  var infile = ""
  var outfile = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/default.tsv"
  var model = ""
  var minSup = 0.30

}
```

H.1.13 ExtractThings.scala

```
package ca.mikelavender.CommunityCompetition.speciesrecoverytests

import java.io.File
import scala.io.Source
import scala.collection.immutable.Iterable

/**
 * Created with IntelliJ IDEA.
 * User: MikeLavender
 * Date: 12/31/2013
 * Time: 9:54 AM
 *
 * This class pulls the PA matrix out of the run files generated by some other method
 */

class ExtractThings {

  def getMatrix(infile: File): Array[Array[Int]] = {
    for (line <- Source.fromFile(infile).getLines() if line.contains(":\t")) yield {
      line.split(":\t")(1).split(",").map(_.trim.toInt)
    }
  }.toArray

  def getFirstLine(infile: File): Set[String] = {
    val src = Source.fromFile(infile)
    val firstLine = src
      .getLines()
      .take(1)
      .mkString
      .replaceAll( """List\(|\)""", "")
      .split(", ")
      .toSet // I want to convert the list of species to a Matrix
    src.close()

    firstLine
  }

  /**
   * Takes a subset results file and extracts the significane (gt) subsets and groups them into a list of significant
   * subsets.
```

```
   * @param infile is a file consisting of subset tests for a specific matrix
   * @return A List of Tuples were the first element is the size of the subset and the second element is
   *       the list of subsets that are significant (gt)
   */
  def getSubsets(infile: File): List[(Int, List[List[Int]])] = groupIterator(readFile(infile))

  def getSubsetsItemSets(infile: File): Iterable[List[(String, Array[Int])]] =
groupIteratorItemSet(readFileItemSet(infile))

  private def readFile(infile: File): List[List[Int]] = {
    var headerMap = Map[String, Int]()
    var parsedFile = List[List[Int]]()
    val lines = Source.fromFile(infile).getLines()

    var splitLength = 0
    var headerKey = ""

    Config.model match {
      case "co" => {
        splitLength = 6
        headerKey = "CScore"
      }
      case "nn" => {
        splitLength = 8
        headerKey = "AITS_NN"
      }
      case "sdnn" => {
        splitLength = 8
        headerKey = "AITS_SDNN"
      }
    }

    //todo: to length needs to be adjusted on a per file (test) basis
    for (line <- lines; if (line split ("\t")).length == splitLength) {
      if (line.contains("Sp_Group")) {
        headerMap = buildHeaderMap(line)
      } else {
        val currentLine = line split ("\t")
        //todo: the column to look in is hard coded but should be a parameter passed to the method
        if (currentLine(headerMap(headerKey)).contains("gt")) {
          parsedFile = extractToList(currentLine(headerMap("Sp_Group"))) :: parsedFile
        }
      }
    }

    /**
     * Builds a lookup table of column header to array index number. This allows
     * us to get values by using the header name instead of the column # -1
     * @param line  the line from the text file that contains the header values
     *          separated by <tab> characters
     * @return     Map(String, Int)
     **/
    def buildHeaderMap(line: String) = line.split("\t").zipWithIndex.toMap
```

```
  /**
   *
   * @param s
   * @return
   */
  def extractToList(s: String): List[Int] = {
   if (s == "List()") List[Int]()
   else {
    s.replaceAll( """(List\(|\)| )""", "")
     .split(",")
     .toList
     .map(_.toInt)
     .sorted
   }
  }
  parsedFile
 }

 private def readFileItemSet(infile: File): List[(String, Array[Int])] = {
  var hMap = Map[String, Int]()
  var parsedFile = List[Tuple2[String, Array[Int]]]()
  val lines = Source.fromFile(infile).getLines()

  var splitLength = 0
  var headerKey = ""

  Config.model match {
   case "co" => {
    splitLength = 6
    headerKey = "CScore"
   }
   case "nn" => {
    splitLength = 8
    headerKey = "AITS_NN"
   }
   case "sdnn" => {
    splitLength = 8
    headerKey = "AITS_SDNN"
   }
  }


  for (line <- lines; if (line split "\t").length == splitLength) {
   if (line.contains("Sp_Group")) {
    hMap = buildMap(line)
   } else {
    val currentLine = line split "\t"
    parsedFile = (currentLine(hMap(headerKey)).replaceAll( """>""", ""),
extractToList(currentLine(hMap("Sp_Group")))) :: parsedFile
   }
  }
```

```
    def buildMap(line: String) = line.split("\t").zipWithIndex.toMap

    def extractToList(s: String): Array[Int] = {
      if (s == "List()") Array[Int]()
      else {
        s.replaceAll( """(List\(|\)| )""", "")
          .split(",")
          .map(_.toInt)
      }
    }
    parsedFile
  }

  private def groupIterator(parsedFile: List[List[Int]]): List[(Int, List[List[Int]])] = {
    parsedFile
      .groupBy(f => f.length)
      .toList
  }

  private def groupIteratorItemSet(parsedFile: List[(String, Array[Int])]): Iterable[List[(String, Array[Int])]] = {
    parsedFile
      .groupBy(f => f._2.length)
      .map(_._2)
  }

}
```

H.1.14 Logger.scala

```
package ca.mikelavender.CommunityCompetition.speciesrecoverytests

 import java.io.FileWriter

 /**
  * Created with IntelliJ IDEA.
  * User: MikeLavender
  * Date: 2014-03-20
  * Time: 5:16 PM
  */
 object Logger {
  var appendFlag = false
  val header = "sp\tpl\tn\tss\tmatch\tmissmatch\ttest\tfile"
  private val config = Config

  writeToFile(header + "\n")
  println(header)
  appendFlag = true

  def writeToFile(string: String) {
    val out = new FileWriter(config.outfile, appendFlag)
    try {
```

```
      out.write(string + "\n")
    } finally {
      out.close()
    }
  }

}
```

H.1.15 Runner.scala

```
package ca.mikelavender.CommunityCompetition.speciesrecoverytests

import ca.mikelavender.CommunityCompetition.cooccurrence.pairwise.veech.VeechPairwise
import ca.mikelavender.CommunityCompetition.cooccurrence.pairwise.clcriterion.CLCriterion
import ca.mikelavender.CommunityCompetition.indicatorspecies.IndVal
import ca.mikelavender.CommunityCompetition.frequentitemsets.fpgrowth.FPRunner
import java.io.File
import scala.actors.Future
import scala.actors.Futures._
import ca.mikelavender.CommunityCompetition.common.FactorialMap


/**
 * Created with IntelliJ IDEA.
 * User: MikeLavender
 * Date: 2014-03-19
 * Time: 4:55 PM
 */
object Runner {
  def main(args: Array[String]) {
    val runner = new Runner

    val usage =
      """
        | Usage: java -jar CompetitionTests.jar [--mtxFile <PATH>] [--traitFile <PATH>] [--nullCount num] [--subsets
num]
        |    [--type <is|fp|veech|cl>] => the type of analysis to run. Defaults to 'fp'
        |    [--path <PATH>] => path to directory, file or zip file to process.
        |    [--model <co|nn|sdnn>] => null model being tested.
        |    usage => prints this message
        |
        |    Examples: java -jar CompetitionTests.jar blah blah blah
      """.stripMargin

    if (args.contains("usage")) {
      println(usage)
      sys.exit(1)
    }
    val argList = args.toList
    type OptionMap = Map[Symbol, Any]
```

```
  def nextOption(map: OptionMap, list: List[String]): OptionMap = {
    list match {
      case Nil => map
      case "--type" :: value :: tail => nextOption(map ++ Map('type -> value.toString), tail)
      case "--minSup" :: value :: tail => nextOption(map ++ Map('minSup -> value.toInt), tail)
      case "--infile" :: value :: tail => nextOption(map ++ Map('infile -> value.toString), tail)
      case "--outfile" :: value :: tail => nextOption(map ++ Map('outfile -> value.toString), tail)
      case "--model" :: value :: tail => nextOption(map ++ Map('model -> value.toString), tail)
      case option :: tail => println("Unknown option " + option)
        sys.exit(1)
    }
  }
  val options = nextOption(Map(), argList)

  runner.run(options)

 }
}

class Runner {
 private val config = Config
 private var analysis = ""

 def run(options: Map[Symbol, Any]) {

  config.typ = options.getOrElse('type, "").asInstanceOf[String]
  config.minSup = options.getOrElse('minSup, 30).asInstanceOf[Int].toDouble / 100
  config.infile = options.getOrElse('infile, "").asInstanceOf[String]
  config.outfile = options.getOrElse('outfile, config.model + "_" + config.typ + ".tsv").asInstanceOf[String]
  config.model = options.getOrElse('model, "").asInstanceOf[String]

  val out = Logger

  println("minSup = " + config.minSup)

  val testType = config.typ
  val path = config.infile

  // this is a hack to stop exceptions in the veech method when it first starts up
  if (testType == "veech") (2 to 1000).foreach(i => FactorialMap.build(i))

  // Set things up to run concurrently (use all cores available)
  var tasks = List[Future[Unit]]()

  for (file <- fileIterator(path)) {
    val f = future {
      processingLoop(file, testType)
    }
    tasks = f :: tasks
  }
  tasks.map(f => f())
 }
```

```
  def processingLoop(file: File, testType: String) {
    val extractor = new ExtractThings
    val regex =
"""sp(5|10|15|20|40|60|80|100)p(20|25|50|75|40|60|80|100|150|200)perm100000N(0|5|10|15|20)Type.*
\.log""".r
    val regex(sp, pl, n) = file.getName

    val expectedSet = extractor.getFirstLine(file) - ""
    val matrix = extractor.getMatrix(file)

    testType match {
      case "fp" => logResultSet(expectedSet.map(_.toInt), fpRunner(file), sp, pl, n, file)
      case "is" => logResultSet(expectedSet.map(_.toInt), isRunner(file), sp, pl, n, file)
      case "veech" => logResultSet(expectedSet.map(_.toInt), veechRunner(matrix), sp, pl, n, file)
      case "cl" => logResultSet(expectedSet.map(_.toInt), clRunner(matrix), sp, pl, n, file)
    }
  }

  def logResultSet(expectedSet: Set[Int], resultSet: List[(Int, Set[Int])], sp: String, pl: String, n: String, file: File) {
    for (r <- resultSet) {
      // get the intersection between expected & observed (match)
      val Match = expectedSet.intersect(r._2).size

      // get all the elements in the result set not in the expected set (miss-match)
      val missMatch = r._2.diff(expectedSet).size

      val logEntry =
        sp + "\t" +
          pl + "\t" +
          n + "\t" +
          r._1 + "\t" +
          Match + "\t" +
          missMatch + "\t" +
          analysis + "\t" +
          file.getName

      println(logEntry)
      Logger.writeToFile(logEntry)

    }
  }

  def logResultSet(expectedSet: Set[Int], resultSet: Set[Int], sp: String, pl: String, n: String, file: File) {
    val Match = expectedSet.intersect(resultSet).size

    // get all the elements in the result set not in the expected set (miss-match)
    val missMatch = resultSet.diff(expectedSet).size

    val logEntry =
      sp + "\t" +
        pl + "\t" +
        n + "\t" +
```

```
    "NA" + "\t" +
    Match + "\t" +
    missMatch + "\t" +
    analysis + "\t" +
    file.getName

  println(logEntry)
  Logger.writeToFile(logEntry)

}

def fpRunner(file: File): List[(Int, Set[Int])] = {
  val extractor = new ExtractThings
  analysis = "fpSet" + (config.minSup * 100).toInt
  for (g <- extractor.getSubsets(file).filter(group => group._2.length >= 50)) yield
    (g._2.head.length,
      new FPRunner(g._2.map(_.map(_.toString)))
        .process
        .filter(_._1.length == 2).view
        .map(_._1.toSet)
        .foldLeft(Set[Int]())((acc, x) => acc ++ x.map(_.toInt)))
}

def isRunner(file: File): List[(Int, Set[Int])] = {
  val extractor = new ExtractThings
  analysis = "indVal"
  for (g <- extractor.getSubsetsItemSets(file)) yield
    (g.head._2.length, (new IndVal).process(g).filter(_._2._3 > 0.95).view
      .foldLeft(Set[Int]())((acc, x) => acc + x._1))
}.toList

def veechRunner(matrix: Array[Array[Int]]): Set[Int] = {
  analysis = "veech"
  (new VeechPairwise)
    .getSetOfAllPairwise(matrix)
    .filter(p => (p._2._2 + p._2._3) < 0.05).view
    .foldLeft(Set[Int]())((acc, x) => acc +(x._1._1, x._1._2))
}

def clRunner(matrix: Array[Array[Int]]): Set[Int] = {
  analysis = "95CL"
  (new CLCriterion)
    .getAllPairwiseNullModel(matrix, 200) // more than 200 and we have to allocate more memory
    .filter(p => (p._2._3) > 0.95).view
    .foldLeft(Set[Int]())((acc, x) => acc +(x._1._1, x._1._2))
}

def fileIterator(path: String): Iterator[File] = {
  new File(path)
    .listFiles.filter(f => f.isFile
    & f.getName.endsWith(".log")
    ).toIterator
}
```

```
}
```

H.1.16 BuildCOTestMatrices.scala

```
package ca.mikelavender.CommunityCompetition.subsettests

import scala.util.Random
import ca.mikelavender.nullmodeller.{NullModels, MatrixRandomisation, MatrixStatistics}
import ca.mikelavender.CommunityCompetition.matrixFactory.MatrixFactory

object BuildCOTestMatrices {

  var paMatrix: Array[Array[Int]] = _
  var tMatrix: Array[Array[Double]] = _
  var maxSpecies: List[Int] = _
  var subSetNullResult: List[Double] = _


  def setup(species: Int, plots: Int, n: Int, typ: String = "nn", permMax: Int = 10000) {

    val mtxFactory = new MatrixFactory

    /**
     * 1. Create a PA matrix of dimension Sp x Pl.
     * 6. Write PA matrix
     * 8. Write maximized species
     */

    var pa = Array.empty[Array[Int]]

    pa = mtxFactory.buildMatrix(species, plots)

    val maximal: List[Int] = {
      var maximizedC = Tuple3(false, List[Int](), List[Double]())
      while (maximizedC._1 != true) {
        // use a cloned matrix so that we don't make changes to the original
        maximizedC = getMaximalCScore(pa.map(_.clone()), permMax, n)
      }
      subSetNullResult = maximizedC._3
      maximizedC._2
    }

    maxSpecies = maximal

  }

  def getMaximalCScore(a1: Array[Array[Int]], permMax: Int, n: Int): (Boolean, List[Int], List[Double]) = {
    val stats = new MatrixStatistics
    val shuffler = new MatrixRandomisation
```

```
  var changed = false
  var maxdSpecies = List[Int]()
  var subNull = List[Double]()

  if (n != 0 & n != 1) {

    val sp2maximize: List[Int] = randomSelect(n, a1.indices.toList).sorted

    val subArray = sp2maximize.map(f => a1(f)).toArray

    var maxC = stats.cScore(subArray, n, subArray(0).length, false)
    var maxArray = Array.fill(n, a1(0).length)(0)

    var loop = 0

    while (loop <= permMax) {
      loop += 1
      shuffler.coOccFixedEquiProb(subArray)
      val tempC = stats.cScore(subArray, n, subArray(0).length, false)
      if (tempC > maxC) {
        maxC = tempC
        maxArray = subArray.map(_.clone())
        changed = true
      }
    }

    for (i <- 0 to n - 1) {

      a1(sp2maximize(i)) = maxArray(i)
    }
    maxdSpecies = sp2maximize ::: maxdSpecies

    if (changed == true) {
      val nm = new NullModels
      subNull = nm.iSwapNullAnalysisController(maxArray, 5000, 1, normalized = false).reverse
    }

  } else changed = true

  paMatrix = a1
  (changed, maxdSpecies.sorted, subNull)
}

def removeAt[A](n: Int, xs: List[A]): (List[A], A) = {
  val (heads, tails) = xs.splitAt(n)
  (heads ::: tails.tail, tails.head)
}

def randomSelect[A](n: Int, xs: List[A]): List[A] = (n, xs) match {
  case (_, Nil) => Nil
  case (0, _) => Nil
  case (_, _) => {
    val x = removeAt(new Random().nextInt(xs.size), xs)
```

```
      x._2 :: randomSelect(n - 1, x._1)
    }
  }

  def isSignificant(obs: Double, exp: List[Double]) = {
    val lt = exp.count(_ > obs).toDouble / exp.length.toDouble
    val gt = exp.count(_ < obs).toDouble / exp.length.toDouble
    var sig = "ns"

    if (gt >= 0.975) sig = "gt"
    if (lt >= 0.975) sig = "lt"

    if (gt >= 0.995) sig = "gt>"
    if (lt >= 0.995) sig = "lt>"

    if (gt >= 0.9995) sig = "gt>>"
    if (lt >= 0.9995) sig = "lt>>"

    sig
  }
}
```

H.1.17 BuildLSTestMatrices.scala

```
package ca.mikelavender.CommunityCompetition.subsettests

import util.Random
import ca.mikelavender.nullmodeller.{MatrixRandomisation, MatrixStatistics}
import collection.mutable.ArrayBuffer
import collection.immutable.IndexedSeq
import ca.mikelavender.CommunityCompetition.matrixFactory.MatrixFactory

object BuildLSTestMatrices {

  var paMatrix: Array[Array[Int]] = _
  var tMatrix: Array[Array[Double]] = _
  var maxSpecies: List[Int] = _


  def setup(species: Int, plots: Int, n: Int, typ: String = "ntd", permMax: Int = 10000) {

    val mtxFactory = new MatrixFactory

    /**
     * 1. Create a PA matrix of dimension Sp x Pl.
     * 2. Create an Array of Sp traits.
     * 3. Determine all -ve co-occurring species.
     * 4. Pick N species (not -ve co-occurring) and maximize NTD/SDNN
     * 5. Assign trait values to remaining species randomly.
     * 6. Write PA matrix
     * 7. Write trait matrix
```

```
    * 8. Write maximized species
    */

   var pa = Array.empty[Array[Int]]
   var nonNegOccurringSp = List[Int]()

   do {
    // generate PA matrix
    pa = mtxFactory.buildMatrix(species, plots)

    // get -ve occurring species
    val negOccurringSp = getAllPairwiseVeech(pa).filter(p => p._3._2 + p._3._1 <= 0.05).groupBy(f =>
f._1).map(_._1).toSet

    // get non -ve occurring species
    nonNegOccurringSp = (0 to species - 1).toSet.diff(negOccurringSp).toList

    // the logic here is if N = 0 or one then we don't care about +/- co-occurrence - it is just random
   } while (n > 1 & nonNegOccurringSp.length < n)
   // generate trait values
   val traits = Array.fill(species)(math.abs((Random.nextDouble() + 0.001) * 10000).toInt / 100d)

   val maximal: Map[Int, Double] = {
    typ match {
     case "ntd" => getMaximalNTD(nonNegOccurringSp, traits, pa, permMax, n)
     case "sdnn" => getMinimalSDNN(nonNegOccurringSp, traits, pa, permMax, n)
     case "co" => {
      var maximizedC = false
      while (maximizedC != true) {
       maximizedC = getMaximalCScore(pa.map(_.clone()), permMax, n)
      }
      Map()
     }
     case _ => sys.error("No \"type\" provided in command line.")
    }
   }

   // return a Map of species -> traits
   val allAssigned: Map[Int, Double] = assignTraitsToMatrix(pa, traits, maximal)

   paMatrix = pa
   val tempMatrix = Array.fill(species, 1)(0d)
   allAssigned.map(x => tempMatrix(x._1)(0) = x._2)
   tMatrix = tempMatrix
   maxSpecies = maximal.map(_._1).toList.sorted

  }

  def assignTraitsToMatrix(pa: Array[Array[Int]], traits: Array[Double], maximal: Map[Int, Double]): Map[Int,
Double] = {
   getRemainders(traits, maximal, pa.length) ++ maximal
  }
```

```
  def getMaximalNTD(nonNegOccurringSp: List[Int], traits: Array[Double], pa: Array[Array[Int]], permMax: Int, n:
Int): Map[Int, Double] = {

    val mtxStats = new MatrixStatistics

    var maximal: Map[Int, Double] = Map()
    var maxNTD = 0d

    // start maximise mNTD loop (100000 times or more)
    var loop = 0
    while (loop < permMax & n > 1) {

      // randomly choose N species indices from non -ve occurring
      val choosenSpecies = randomSelect(n, nonNegOccurringSp)

      // randomly choose N trait values
      val choosenTraits = randomSelect(n, traits.toList)

      // create trait sub matrix and store species -> trait Map
      var map: Map[Int, Double] = Map()
      var traitCounter = 0
      val traitSubMatrix = Array.fill(pa.length, pa(0).length)(0d)
      for (r <- choosenSpecies) {
        traitSubMatrix(r) = pa(r).map(_ * choosenTraits(traitCounter))
        map += (r -> choosenTraits(traitCounter))
        traitCounter += 1
      }

      // calc mNTD
      val currentNTD = mtxStats.meanAndStdDevNTD(traitSubMatrix
        .filter(p => p.sum != 0)
        .transpose
        .filter(p => p.sum != 0)
        .transpose)

      // check to see if it is larger than the last or 0
      if (currentNTD._1 > maxNTD | maxNTD == 0) {
        maxNTD = currentNTD._1
        maximal = map
      }

      loop += 1
    }
    maximal
  }

  def getMinimalSDNN(nonNegOccurringSp: List[Int], traits: Array[Double], pa: Array[Array[Int]], permMax: Int, n:
Int): Map[Int, Double] = {

    val mtxStats = new MatrixStatistics

    var maximal: Map[Int, Double] = Map()
    var maxNTD = Double.MaxValue
```

```
//    var maxNTD = 0d

// start maximise mNTD loop (100000 times or more)
var loop = 0
while (loop < permMax & n > 1) {

  // randomly choose N species indices from non -ve occurring
  val choosenSpecies = randomSelect(n, nonNegOccurringSp)

  // randomly choose N trait values
  val choosenTraits = randomSelect(n, traits.toList)

  // create trait sub matrix and store species -> trait Map
  var map: Map[Int, Double] = Map()
  var traitCounter = 0
  val traitSubMatrix = Array.fill(pa.length, pa(0).length)(0d)
  for (r <- choosenSpecies) {
    traitSubMatrix(r) = pa(r).map(_ * choosenTraits(traitCounter))
    map += (r -> choosenTraits(traitCounter))
    traitCounter += 1
  }

  // calc mNTD
  val currentNTD = mtxStats.meanAndStdDevNTD(traitSubMatrix
    .filter(p => p.sum != 0)
    .transpose
    .filter(p => p.sum != 0)
    .transpose)

  // check to see if it is larger than the last or 0
  if (currentNTD._2 < maxNTD | maxNTD == Double.MaxValue) {
    //     if (currentNTD._2 > maxNTD | maxNTD == 0) {
    maxNTD = currentNTD._2
    maximal = map
  }

  loop += 1
 }
 maximal
}

def getMaximalCScore(a1: Array[Array[Int]], permMax: Int, n: Int) = {
 val stats = new MatrixStatistics
 val shuffler = new MatrixRandomisation

 var changed = false

 if (n != 0 & n != 1) {

  val sp2maximize: List[Int] = randomSelect(n, a1.indices.toList).sorted

  val subArray = sp2maximize.map(f => a1(f)).toArray
```

```
    var maxC = stats.cScore(subArray, n, subArray(0).length, false)
    var maxArray = Array.fill(n, a1(0).length)(0)

    var loop = 0

    while (loop <= permMax) {
     loop += 1
     shuffler.coOccFixedEquiProb(subArray)
     val tempC = stats.cScore(subArray, n, subArray(0).length, false)
     if (tempC > maxC) {
      maxC = tempC
      maxArray = subArray.map(_.clone())
      changed = true
     }
    }

    for (i <- 0 to n - 1) {
     a1(sp2maximize(i)) = maxArray(i)
    }

   } else changed = true

   paMatrix = a1
   changed
  }


  def getRemainders(allTraits: Array[Double], traitMap: Map[Int, Double], paSize: Int): Map[Int, Double] = {
   val traits = traitMap.map(_._2).toArray
   val species = traitMap.map(_._1).toArray
   val unusedTraits = allTraits.diff(traits)
   val traitlessSpecies = (0 to paSize - 1).toArray.diff(species)
   if (unusedTraits.size != traitlessSpecies.size) {
    sys.error("Oh Shit!!!\n" +
     "all traits: " + allTraits.sorted.mkString(", ") + "\n" +
     "unused traits: " + unusedTraits.toArray.sorted.mkString(", ") + "\n" +
     "species: " + traitMap.map(_._1).toArray.sorted.mkString(", ") + "\n" +
     "traitless species: " + traitlessSpecies.toArray.sorted.mkString(", ") + "\n")
   } else
    Map(shuffle(traitlessSpecies.toArray).zip(shuffle(unusedTraits.toArray)).toArray: _*)
  }

  def getAllPairwiseVeech(matrix: Array[Array[Int]]): IndexedSeq[(Int, Int, (Double, Double, Double))] = {
   val mtxStats = new MatrixStatistics
   for (x <- 0 to matrix.length - 1; y <- 0 to matrix.length - 1)
   yield (x, y, mtxStats.veech(matrix(x), matrix(y)))
  }

  def removeAt[A](n: Int, xs: List[A]): (List[A], A) = {
   val (heads, tails) = xs.splitAt(n)
   (heads ::: tails.tail, tails.head)
  }
```

```
  def randomSelect[A](n: Int, xs: List[A]): List[A] = (n, xs) match {
   case (_, Nil) => Nil
   case (0, _) => Nil
   case (_, _) => {
    val x = removeAt(new Random().nextInt(xs.size), xs)
    x._2 :: randomSelect(n - 1, x._1)
   }
  }

  def shuffle[T](array: Array[T]) = {
   val buf = new ArrayBuffer[T] ++= array

   def swap(i1: Int, i2: Int) {
    val tmp = buf(i1)
    buf(i1) = buf(i2)
    buf(i2) = tmp
   }

   for (n <- buf.length to 2 by -1) {
    val k = Random.nextInt(n)
    swap(n - 1, k)
   }
   buf
  }
}
```

H.1.18 Config.scala

```
package ca.mikelavender.CommunityCompetition


 object Config {

  var mtxFile: String = null
  var traitFile: String = null

  var nullCount = 5000
  var subsets = 100000
  var reps = 1
  var cpu = 1
  var silent = false
  var outfile = "outfile.tsv"


  var normalized = false
  var throttle = 1

  var species = 100
  var plots = 100
  var permNTD = 100000
  var percentStructure = .2
```

```
  var N = 0
  var typ = "nn"
  var alwaysRunCo_occurrence = false


 }
```

H.1.19 FileObjects.scala

```
package ca.mikelavender.CommunityCompetition

 trait FileObjects {

  var header: IndexedSeq[String]

  var labels: IndexedSeq[String]

 }

 trait MatrixObject extends FileObjects {

  var matrix: Array[Array[Int]]

 }

 trait TraitsObject extends FileObjects {

  var matrix: Array[Array[Double]]

 }
```

H.1.20 FileTools.scala

```
package ca.mikelavender.CommunityCompetition

 import io.Source
 import scala.Array._

 class FileTools {

  // Reads a file and stores it in a Matrix object.
  def readPA(file: String, hasHeader: Boolean = false, hasRowLabels: Boolean = false, delimiter: String = ",") = {

    val mtx = new MatrixObject {
      var labels: IndexedSeq[String] = _
      var header: IndexedSeq[String] = _
      var matrix: Array[Array[Int]] = _
    }

    val matrixDimensions = scan(file, delimiter)
```

```
  var sites_col = matrixDimensions._1
  var species_row = matrixDimensions._2

  // Configure the Array
  (hasHeader, hasRowLabels) match {
    case (true, true) => sites_col = sites_col - 1; species_row = species_row - 1
    case (true, false) => species_row = species_row - 1
    case (false, true) => sites_col = sites_col - 1
    case (_, _) => None
  }
  if (hasHeader) print("File has header row. ") else print("File has no header row. ")
  if (hasRowLabels) println("File has row labels.") else println("File has no row labels.")
  println("\tConfigured " + species_row + " x " + sites_col + " matrix.")
  val matrix: Array[Array[Int]] = ofDim(species_row, sites_col)

  val source = Source.fromFile(file)

  var header: IndexedSeq[String] = IndexedSeq()
  var labels: IndexedSeq[String] = IndexedSeq()
  var rowID = 0

  for (line <- source.getLines().filter(p => p.length > 1)) {
    //     val parsed = parse(line, delimiter, hasRowLabels)
    (rowID, hasHeader) match {
      case (0, true) => header = line.split(delimiter).map(_.trim)
      case (_, true) => {
        matrix(rowID - 1) = parseInt(line, delimiter, hasRowLabels)._2
        if (hasRowLabels) labels = labels :+ parseInt(line, delimiter, hasRowLabels)._1
      }
      case (_, false) => {
        matrix(rowID) = parseInt(line, delimiter, hasRowLabels)._2
        if (hasRowLabels) labels = labels :+ parseInt(line, delimiter, hasRowLabels)._1
      }
    }

    rowID += 1
  }
  source.close()
  mtx.header = header
  mtx.labels = labels
  mtx.matrix = matrix

  mtx
}

private def parseInt(line: String, delimiter: String, hasLabels: Boolean): (String, Array[Int]) = {
  val data = line.split(delimiter)
  hasLabels match {
    case true => (data(0), data.drop(1).map(_.trim.toInt))
    case _ => ("", data.map(_.trim.toInt))
  }
}
```

```
// Reads a file and stores it in a Matrix object.
def readTrait(file: String, hasHeader: Boolean = false, hasRowLabels: Boolean = false, delimiter: String = ",") = {

 val traits = new TraitsObject {
  var labels: IndexedSeq[String] = _
  var header: IndexedSeq[String] = _
  var matrix: Array[Array[Double]] = _
 }

 val matrixDimensions = scan(file, delimiter)
 var sites_col = matrixDimensions._1
 var species_row = matrixDimensions._2

 // Configure the Array
 (hasHeader, hasRowLabels) match {
  case (true, true) => sites_col = sites_col - 1; species_row = species_row - 1
  case (true, false) => species_row = species_row - 1
  case (false, true) => sites_col = sites_col - 1
  case (_, _) => None
 }
 if (hasHeader) print("File has header row. ") else print("File has no header row. ")
 if (hasRowLabels) println("File has row labels.") else println("File has no row labels.")
 println("\tConfigured " + species_row + " x " + sites_col + " matrix.")
 val matrix: Array[Array[Double]] = ofDim(species_row, sites_col)

 val source = Source.fromFile(file)

 var header: IndexedSeq[String] = IndexedSeq()
 var labels: IndexedSeq[String] = IndexedSeq()
 var rowID = 0

 for (line <- source.getLines().filter(p => p.length > 1)) {
  (rowID, hasHeader) match {
   case (0, true) => header = line.split(delimiter).map(_.trim)
   case (_, true) => {
    matrix(rowID - 1) = parseDouble(line, delimiter, hasRowLabels)._2
    if (hasRowLabels) labels = labels :+ parseDouble(line, delimiter, hasRowLabels)._1
   }
   case (_, false) => {
    matrix(rowID) = parseDouble(line, delimiter, hasRowLabels)._2
    if (hasRowLabels) labels = labels :+ parseDouble(line, delimiter, hasRowLabels)._1
   }
  }

  rowID += 1
 }
 source.close()
 traits.header = header
 traits.labels = labels
 traits.matrix = matrix

 traits
}
```

```
  private def parseDouble(line: String, delimiter: String, hasLabels: Boolean): (String, Array[Double]) = {
    val data = line.split(delimiter)
    hasLabels match {
      case true => (data(0), data.drop(1).map(_.trim.toDouble))
      case _ => ("", data.map(_.trim.toDouble))
    }
  }

  // Scans a CSV file to try and determine the number of rows and columns.
  private def scan(file: String, delimiter: String) = {
    var sites_col = 0
    var species_row = 0
    println("File: " + file)
    val source = Source.fromFile(file)

    // figure out the number columns in the file given a delimiter
    sites_col = source.getLines().next().split(delimiter).length

    // now get the number of rows but add one on since we used one up figuring out the col's
    species_row = source.getLines().count(s => s.length > 1) + 1

    // Done - close the file
    source.close()
    (sites_col, species_row)
  }
}
```

H.2 Scala scripts

H.2.1 SubSetTests_co.scala

```
import java.io.{FileWriter, BufferedWriter, File}
 import scala.io.Source

 val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Tests/co"

 def getFiles(path: String): Iterator[File] = new File(path)
   .listFiles.filter(f => f.isFile &
   f.getName.endsWith(".log"))
   .toIterator

 def isSignificant(v: String) = if (v.contains("gt")) 1 else 0

 var map = Map[(Int, Int, Int, String, Int), (Int, Int)]()

 for (file <- getFiles(path)) {
   println(file.getName)
```

```
  // parse file name to get sp, p, N and type
  val regex =
"""sp(5|10|15|20|40|60|80|100)p(20|25|50|75|40|60|80|100)perm100000N(0|5|10|15|20)Type-
(nn|sdnn|co)_[0-9]{1,5}_SubsetCo\.v1\.log""".r
  val regex(sp, pl, n, typ) = file.getName
  //  println("sp: " + sp + "\tpl: " + pl + "\tn: " + n + "\ttyp: " + typ)

  val lines = Source.fromFile(file).getLines()

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines; if (line split ("\t")).length == 6) {

    if (hFlag && line.contains("Sp_Group")) {
      hMap = buildMap(line)
      hFlag = false
    } else if (hFlag == false) {
      val currentLine = line split ("\t")
      val subSetSize = currentLine(hMap("SP")).toInt

      if (sp.toInt != 5) {
        if (!map.contains(sp.toInt, pl.toInt, n.toInt, n + "-" + typ + "-" + subSetSize, subSetSize)) {
          map += (sp.toInt, pl.toInt, n.toInt, n + "-" + typ + "-" + subSetSize, subSetSize) -> {
            (1,
              isSignificant(currentLine(hMap("CScore")))
              )
          }
        } else {
          map += (sp.toInt, pl.toInt, n.toInt, n + "-" + typ + "-" + subSetSize, subSetSize) -> {
            (map((sp.toInt, pl.toInt, n.toInt, n + "-" + typ + "-" + subSetSize, subSetSize))._1 + 1,
              map((sp.toInt, pl.toInt, n.toInt, n + "-" + typ + "-" + subSetSize, subSetSize))._2 +
isSignificant(currentLine(hMap("CScore")))
              )
          }
        }
      }
    }
  }
 }



 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
3/Summarized Data/SubSetTests_co.tsv"))
 f.write("Species\tPlots\tN\tType\tSubSize\tCount\tval" + "\n")

 println("\nSpecies\tPlots\tN\tType\tSubSize\tCount\tval")
 for (key <- map.keySet.toList.sorted) {
   println(
     key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + map(key)._1 + "\t" + map(key)._2.toDouble
/ map(key)._1
   )
```

```
  f.write(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + map(key)._1 + "\t" + map(key)._2.toDouble
/ map(key)._1 + "\n"
  )

  f.flush()
 }

 f.flush()
 f.close()


 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

H.2.2 SubSetTests_nn.scala

```
import java.io.{FileWriter, BufferedWriter, File}
 import scala.io.Source

 val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Tests/nn"

 def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

 def isSignificant(v: String) = if (v.contains("gt")) 1 else 0

 var map = Map[(Int, Int, Int, String, Int), (Int, Int)]()

 for (file <- getFiles(path)) {
  println(file.getName)

  // parse file name to get sp, p, N and type
  val regex = """sp(5|10|15|20|40|60|80|100)p(25|50|100|150|200)perm100000N(0|5|10|15|20)Type-
(nn|sdnn|co)_[0-9]{1,5}_Subset\.v1\.log""".r
  val regex(sp, pl, n, typ) = file.getName
  //  println("sp: " + sp + "\tpl: " + pl + "\tn: " + n + "\ttyp: " + typ)

  val lines = Source.fromFile(file).getLines()

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines; if (line split ("\t")).length == 8) {

   if (hFlag && line.contains("Sp_Group")) {
    hMap = buildMap(line)
    hFlag = false
   } else if (hFlag == false) {
```

```
    val currentLine = line split ("\t")
    val subSetSize = currentLine(hMap("SP")).toInt

    if (sp.toInt != 5) {
     if (!map.contains(sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize)) {
      map += (sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize) -> {
       (1,
        isSignificant(currentLine(hMap("AITS_" + typ.toUpperCase)))
        )
      }
     } else {
      map += (sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize) -> {
       (map((sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize))._1 + 1,
        map((sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize))._2 +
isSignificant(currentLine(hMap("AITS_" + typ.toUpperCase)))
        )
      }
     }
    }
   }
  }
 }

 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
3/Summarized Data/SubSetTests_nn.tsv"))
 f.write("Species\tPlots\tN\tType\tSubSize\tCount\tval" + "\n")

 println("\nSpecies\tPlots\tN\tType\tSubSize\tCount\tval")
 for (key <- map.keySet.toList.sorted) {
  println(
   key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + map(key)._1 + "\t" + map(key)._2.toDouble
/ map(key)._1
  )

  f.write(
   key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + map(key)._1 + "\t" + map(key)._2.toDouble
/ map(key)._1 + "\n"
  )

  f.flush()
 }

 f.flush()
 f.close()


 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

H.2.3 SubSetTests_sdnn.scala

```
import java.io.{FileWriter, BufferedWriter, File}
```

```
import scala.io.Source

val path = "/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Tests/sdnn"

def getFiles(path: String): Iterator[File] = new File(path)
  .listFiles.filter(f => f.isFile &
  f.getName.endsWith(".log"))
  .toIterator

def isSignificant(v: String) = if (v.contains("lt")) 1 else 0

var map = Map[(Int, Int, Int, String, Int), (Int, Int)]()

for (file <- getFiles(path)) {
  println(file.getName)

  // parse file name to get sp, p, N and type
  val regex = """sp(5|10|15|20|40|60|80|100)p(25|50|100|150|200)perm100000N(0|5|10|15|20)Type-
(ntd|sdnn|co)_[0-9]{1,5}_Subset\.v2\.log""".r
  val regex(sp, pl, n, typ) = file.getName
  //  println("sp: " + sp + "\tpl: " + pl + "\tn: " + n + "\ttyp: " + typ)

  val lines = Source.fromFile(file).getLines()

  var hFlag = true
  var hMap = Map[String, Int]()

  for (line <- lines; if (line split ("\t")).length == 8) {

    if (hFlag && line.contains("Sp_Group")) {
      hMap = buildMap(line)
      hFlag = false
    } else if (hFlag == false) {
      val currentLine = line split ("\t")
      val subSetSize = currentLine(hMap("SP")).toInt

      if (sp.toInt != 5) {
        if (!map.contains(sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize)) {
          map += (sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize) -> {
            (1,
              isSignificant(currentLine(hMap("AITS_" + typ.toUpperCase)))
              )
          }
        } else {
          map += (sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize) -> {
            (map((sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize))._1 + 1,
              map((sp.toInt, pl.toInt, n.toInt, n +"-" + typ +"-" + subSetSize, subSetSize))._2 +
isSignificant(currentLine(hMap("AITS_" + typ.toUpperCase)))
              )
          }
        }
      }
    }
```

```
  }
 }



 val f = new BufferedWriter(new FileWriter("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
3/Summarized Data/SubSetTests_sdnn.tsv"))
 f.write("Species\tPlots\tN\tType\tSubSize\tCount\tval" + "\n")

 println("\nSpecies\tPlots\tN\tType\tSubSize\tCount\tval")
 for (key <- map.keySet.toList.sorted) {
  println(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + map(key)._1 + "\t" + map(key)._2.toDouble
/ map(key)._1
  )

  f.write(
    key.toString().replaceAll( """[\(\)]""", "").replace(",", "\t") + "\t" + map(key)._1 + "\t" + map(key)._2.toDouble
/ map(key)._1 + "\n"
  )

  f.flush()
 }

 f.flush()
 f.close()


 def buildMap(line: String) = line.split("\t").zipWithIndex.toMap
```

Appendix I Chapter 4.0 R code

I.1.1 'matchRatesFinal_co_cl.R'

```
rm(list = ls(all = TRUE))

library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/co_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))


# First plot - method means for 95CL
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "95CL",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 30

quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "95CL-0-mean",
        "95CL-5-mean",
        "95CL-10-mean",
        "95CL-15-mean",
        "95CL-20-mean"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
```

```
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_cl_means.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Second plot - method sd for 95CL
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "95CL",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 20

quartz(width = 3.0, height = 7.5)
```

```
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "95CL-0-sd",
        "95CL-5-sd",
        "95CL-10-sd",
        "95CL-15-sd",
        "95CL-20-sd"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -0.5, to = 7.5, length = 17)
 # , cuts = 16
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/matchRatesFinal_co_cl_sd.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for 95CL
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "95CL",]
```

```
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 30

quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "95CL-0-mmxbar",
        "95CL-5-mmxbar",
        "95CL-10-mmxbar",
        "95CL-15-mmxbar",
        "95CL-20-mmxbar"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_cl_mmxbar.png",
type ="png",
```

```
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for 95CL
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "95CL",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 20

quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "95CL-0-mmsd",
        "95CL-5-mmsd",
        "95CL-10-mmsd",
        "95CL-15-mmsd",
        "95CL-20-mmsd"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
```

```
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
          , alternating = FALSE
          , cex = 0.7)
          , x = list(tck = c(0.5, 0)
          , rot=90
          , alternating = FALSE
          , labels = c("25", "50", "100", "150", "200")
          , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_cl_mmsd.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.2 'matchRatesFinal_co_fp.R'

```
rm(list = ls(all = TRUE))

library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/co_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.55, lines = 1.2)
colscaledivs <- 30

# First plot - method means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
```

```
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mean","fpSet30-0-10-mean","fpSet30-0-15-mean","fpSet30-0-20-mean","fpSet30-0-25-
 mean","fpSet30-0-50-mean",
        "fpSet30-5-5-mean","fpSet30-5-10-mean","fpSet30-5-15-mean","fpSet30-5-20-mean","fpSet30-5-25-
 mean","fpSet30-5-50-mean",
        "fpSet30-10-5-mean","fpSet30-10-10-mean","fpSet30-10-15-mean","fpSet30-10-20-mean","fpSet30-10-25-
 mean","fpSet30-10-50-mean",
        "fpSet30-15-5-mean","fpSet30-15-10-mean","fpSet30-15-15-mean","fpSet30-15-20-mean","fpSet30-15-25-
 mean","fpSet30-15-50-mean",
        "fpSet30-20-5-mean","fpSet30-20-10-mean","fpSet30-20-15-mean","fpSet30-20-20-mean","fpSet30-20-25-
 mean","fpSet30-20-50-mean"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_fp_means.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
```

```
device = dev.cur()
)



# Second plot - method sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-sd","fpSet30-0-10-sd","fpSet30-0-15-sd","fpSet30-0-20-sd","fpSet30-0-25-sd","fpSet30-0-50-
 sd",
        "fpSet30-5-5-sd","fpSet30-5-10-sd","fpSet30-5-15-sd","fpSet30-5-20-sd","fpSet30-5-25-sd","fpSet30-5-50-
 sd",
        "fpSet30-10-5-sd","fpSet30-10-10-sd","fpSet30-10-15-sd","fpSet30-10-20-sd","fpSet30-10-25-sd","fpSet30-
 10-50-sd",
        "fpSet30-15-5-sd","fpSet30-15-10-sd","fpSet30-15-15-sd","fpSet30-15-20-sd","fpSet30-15-25-sd","fpSet30-
 15-50-sd",
        "fpSet30-20-5-sd","fpSet30-20-10-sd","fpSet30-20-15-sd","fpSet30-20-20-sd","fpSet30-20-25-sd","fpSet30-
 20-50-sd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -0.5, to = 7.5, length = 17)
 # , cuts = 16
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
```

```
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/matchRatesFinal_co_fp_sd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mmxbar","fpSet30-0-10-mmxbar","fpSet30-0-15-mmxbar","fpSet30-0-20-mmxbar","fpSet30-
 0-25-mmxbar","fpSet30-0-50-mmxbar",
        "fpSet30-5-5-mmxbar","fpSet30-5-10-mmxbar","fpSet30-5-15-mmxbar","fpSet30-5-20-mmxbar","fpSet30-
 5-25-mmxbar","fpSet30-5-50-mmxbar",
        "fpSet30-10-5-mmxbar","fpSet30-10-10-mmxbar","fpSet30-10-15-mmxbar","fpSet30-10-20-
 mmxbar","fpSet30-10-25-mmxbar","fpSet30-10-50-mmxbar",
        "fpSet30-15-5-mmxbar","fpSet30-15-10-mmxbar","fpSet30-15-15-mmxbar","fpSet30-15-20-
 mmxbar","fpSet30-15-25-mmxbar","fpSet30-15-50-mmxbar",
        "fpSet30-20-5-mmxbar","fpSet30-20-10-mmxbar","fpSet30-20-15-mmxbar","fpSet30-20-20-
 mmxbar","fpSet30-20-25-mmxbar","fpSet30-20-50-mmxbar"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
```

```
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("25", "50", "100", "150", "200")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_fp_mmxbar.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
         "fpSet30-0-5-mmsd","fpSet30-0-10-mmsd","fpSet30-0-15-mmsd","fpSet30-0-20-mmsd","fpSet30-0-25-
 mmsd","fpSet30-0-50-mmsd",
         "fpSet30-5-5-mmsd","fpSet30-5-10-mmsd","fpSet30-5-15-mmsd","fpSet30-5-20-mmsd","fpSet30-5-25-
 mmsd","fpSet30-5-50-mmsd",
         "fpSet30-10-5-mmsd","fpSet30-10-10-mmsd","fpSet30-10-15-mmsd","fpSet30-10-20-mmsd","fpSet30-10-
```

```
 25-mmsd","fpSet30-10-50-mmsd",
        "fpSet30-15-5-mmsd","fpSet30-15-10-mmsd","fpSet30-15-15-mmsd","fpSet30-15-20-mmsd","fpSet30-15-
 25-mmsd","fpSet30-15-50-mmsd",
        "fpSet30-20-5-mmsd","fpSet30-20-10-mmsd","fpSet30-20-15-mmsd","fpSet30-20-20-mmsd","fpSet30-20-
 25-mmsd","fpSet30-20-50-mmsd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.15, y = 0.15)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_fp_mmsd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.3 'matchRatesFinal_co_is.scala'

```
rm(list = ls(all = TRUE))
```

```
library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/co_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))


# First plot - method means for is
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 30

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test,
 levels = c(
        "indVal-0-5-mean","indVal-0-10-mean","indVal-0-15-mean","indVal-0-20-mean","indVal-0-25-
 mean","indVal-0-50-mean",
        "indVal-5-5-mean","indVal-5-10-mean","indVal-5-15-mean","indVal-5-20-mean","indVal-5-25-
 mean","indVal-5-50-mean",
        "indVal-10-5-mean","indVal-10-10-mean","indVal-10-15-mean","indVal-10-20-mean","indVal-10-25-
 mean","indVal-10-50-mean",
        "indVal-15-5-mean","indVal-15-10-mean","indVal-15-15-mean","indVal-15-20-mean","indVal-15-25-
 mean","indVal-15-50-mean",
        "indVal-20-5-mean","indVal-20-10-mean","indVal-20-15-mean","indVal-20-20-mean","indVal-20-25-
 mean","indVal-20-50-mean"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
```

```
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("25", "50", "100", "150", "200")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_is_means.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Second plot - method sd for is
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 20

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
         "indVal-0-5-sd","indVal-0-10-sd","indVal-0-15-sd","indVal-0-20-sd","indVal-0-25-sd","indVal-0-50-sd",
         "indVal-5-5-sd","indVal-5-10-sd","indVal-5-15-sd","indVal-5-20-sd","indVal-5-25-sd","indVal-5-50-sd",
```

```
        "indVal-10-5-sd","indVal-10-10-sd","indVal-10-15-sd","indVal-10-20-sd","indVal-10-25-sd","indVal-10-50-
sd",
        "indVal-15-5-sd","indVal-15-10-sd","indVal-15-15-sd","indVal-15-20-sd","indVal-15-25-sd","indVal-15-50-
sd",
        "indVal-20-5-sd","indVal-20-10-sd","indVal-20-15-sd","indVal-20-20-sd","indVal-20-25-sd","indVal-20-50-
sd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -0.5, to = 7.5, length = 17)
 # , cuts = 16
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/matchRatesFinal_co_is_sd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for is
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")
```

```
agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 30

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mmxbar","indVal-0-10-mmxbar","indVal-0-15-mmxbar","indVal-0-20-mmxbar","indVal-0-25-
 mmxbar","indVal-0-50-mmxbar",
        "indVal-5-5-mmxbar","indVal-5-10-mmxbar","indVal-5-15-mmxbar","indVal-5-20-mmxbar","indVal-5-25-
 mmxbar","indVal-5-50-mmxbar",
        "indVal-10-5-mmxbar","indVal-10-10-mmxbar","indVal-10-15-mmxbar","indVal-10-20-mmxbar","indVal-10-
 25-mmxbar","indVal-10-50-mmxbar",
        "indVal-15-5-mmxbar","indVal-15-10-mmxbar","indVal-15-15-mmxbar","indVal-15-20-mmxbar","indVal-15-
 25-mmxbar","indVal-15-50-mmxbar",
        "indVal-20-5-mmxbar","indVal-20-10-mmxbar","indVal-20-15-mmxbar","indVal-20-20-mmxbar","indVal-20-
 25-mmxbar","indVal-20-50-mmxbar"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)
```

```
quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_is_mmxbar.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for is
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 20

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mmsd","indVal-0-10-mmsd","indVal-0-15-mmsd","indVal-0-20-mmsd","indVal-0-25-
 mmsd","indVal-0-50-mmsd",
        "indVal-5-5-mmsd","indVal-5-10-mmsd","indVal-5-15-mmsd","indVal-5-20-mmsd","indVal-5-25-
 mmsd","indVal-5-50-mmsd",
        "indVal-10-5-mmsd","indVal-10-10-mmsd","indVal-10-15-mmsd","indVal-10-20-mmsd","indVal-10-25-
 mmsd","indVal-10-50-mmsd",
        "indVal-15-5-mmsd","indVal-15-10-mmsd","indVal-15-15-mmsd","indVal-15-20-mmsd","indVal-15-25-
 mmsd","indVal-15-50-mmsd",
        "indVal-20-5-mmsd","indVal-20-10-mmsd","indVal-20-15-mmsd","indVal-20-20-mmsd","indVal-20-25-
 mmsd","indVal-20-50-mmsd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = FALSE
 , par.strip.text=p.strip
```

```
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("25", "50", "100", "150", "200")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_is_mmsd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.4 'matchRatesFinal_co_veech.R'

```
rm(list = ls(all = TRUE))

library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/co_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))


# First plot - method means for veech
```

```
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "veech",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 30

quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "veech-0-mean",
        "veech-5-mean",
        "veech-10-mean",
        "veech-15-mean",
        "veech-20-mean"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)
```

```
quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_veech_means.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Second plot - method sd for veech
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "veech",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 20

quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "veech-0-sd",
        "veech-5-sd",
        "veech-10-sd",
        "veech-15-sd",
        "veech-20-sd"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -0.5, to = 7.5, length = 17)
 # , cuts = 16
 , col.regions = (col = rgb.palette(colscaledivs + 1))
```

```
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_veech_sd.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for veech
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "veech",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 30

quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "veech-0-mmxbar",
        "veech-5-mmxbar",
        "veech-10-mmxbar",
        "veech-15-mmxbar",
        "veech-20-mmxbar"
        )
 , labels = c(
        "N=0",
        "N=5",
```

```
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_veech_mmxbar.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for veech
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "veech",]
co$test <- paste(co$test, "-", co$n, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.6, lines = 1.2)
colscaledivs <- 20
```

```
quartz(width = 3.0, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "veech-0-mmsd",
        "veech-5-mmsd",
        "veech-10-mmsd",
        "veech-15-mmsd",
        "veech-20-mmsd"
        )
 , labels = c(
        "N=0",
        "N=5",
        "N=10",
        "N=15",
        "N=20"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(1, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_co_veech_mmsd.png",
type ="png",
width = 3.0,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.5 'matchRatesFinal_nn_fp.R'

```
rm(list = ls(all = TRUE))

library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
  # 4 x method means
  # 4 x miss match means
  # 4 x method sd
  # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/nn_.tsv",
  header = TRUE, sep = "\t", na.strings = c("NA"))

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.55, lines = 1.2)
colscaledivs <- 30

# First plot - method means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
  , levels = c(
        "fpSet30-0-5-mean","fpSet30-0-10-mean","fpSet30-0-15-mean","fpSet30-0-20-mean","fpSet30-0-25-
  mean","fpSet30-0-50-mean",
        "fpSet30-5-5-mean","fpSet30-5-10-mean","fpSet30-5-15-mean","fpSet30-5-20-mean","fpSet30-5-25-
  mean","fpSet30-5-50-mean",
        "fpSet30-10-5-mean","fpSet30-10-10-mean","fpSet30-10-15-mean","fpSet30-10-20-mean","fpSet30-10-25-
  mean","fpSet30-10-50-mean",
        "fpSet30-15-5-mean","fpSet30-15-10-mean","fpSet30-15-15-mean","fpSet30-15-20-mean","fpSet30-15-25-
  mean","fpSet30-15-50-mean",
        "fpSet30-20-5-mean","fpSet30-20-10-mean","fpSet30-20-15-mean","fpSet30-20-20-mean","fpSet30-20-25-
  mean","fpSet30-20-50-mean"
        )
  , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
```

```
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_nn_fp_means.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Second plot - method sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-sd","fpSet30-0-10-sd","fpSet30-0-15-sd","fpSet30-0-20-sd","fpSet30-0-25-sd","fpSet30-0-50-
 sd",
        "fpSet30-5-5-sd","fpSet30-5-10-sd","fpSet30-5-15-sd","fpSet30-5-20-sd","fpSet30-5-25-sd","fpSet30-5-50-
```

```
sd",
        "fpSet30-10-5-sd","fpSet30-10-10-sd","fpSet30-10-15-sd","fpSet30-10-20-sd","fpSet30-10-25-sd","fpSet30-
10-50-sd",
        "fpSet30-15-5-sd","fpSet30-15-10-sd","fpSet30-15-15-sd","fpSet30-15-20-sd","fpSet30-15-25-sd","fpSet30-
15-50-sd",
        "fpSet30-20-5-sd","fpSet30-20-10-sd","fpSet30-20-15-sd","fpSet30-20-20-sd","fpSet30-20-25-sd","fpSet30-
20-50-sd"
        )
, labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
),
, pretty = TRUE
, par.strip.text=p.strip
, strip = strip.custom(bg = "transparent")
, contour = FALSE
, data = ag_co
, region = TRUE
# , at = seq(from = -0.5, to = 7.5, length = 17)
# , cuts = 16
, col.regions = (col = rgb.palette(colscaledivs + 1))
, xlab = "Number of plots"
, ylab = "Number of species"
, layout = c(6, 5)
, scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/matchRatesFinal_nn_fp_sd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
```

```
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mmxbar","fpSet30-0-10-mmxbar","fpSet30-0-15-mmxbar","fpSet30-0-20-mmxbar","fpSet30-
 0-25-mmxbar","fpSet30-0-50-mmxbar",
        "fpSet30-5-5-mmxbar","fpSet30-5-10-mmxbar","fpSet30-5-15-mmxbar","fpSet30-5-20-mmxbar","fpSet30-
 5-25-mmxbar","fpSet30-5-50-mmxbar",
        "fpSet30-10-5-mmxbar","fpSet30-10-10-mmxbar","fpSet30-10-15-mmxbar","fpSet30-10-20-
 mmxbar","fpSet30-10-25-mmxbar","fpSet30-10-50-mmxbar",
        "fpSet30-15-5-mmxbar","fpSet30-15-10-mmxbar","fpSet30-15-15-mmxbar","fpSet30-15-20-
 mmxbar","fpSet30-15-25-mmxbar","fpSet30-15-50-mmxbar",
        "fpSet30-20-5-mmxbar","fpSet30-20-10-mmxbar","fpSet30-20-15-mmxbar","fpSet30-20-20-
 mmxbar","fpSet30-20-25-mmxbar","fpSet30-20-50-mmxbar"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_nn_fp_mmxbar.png",
```

```
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mmsd","fpSet30-0-10-mmsd","fpSet30-0-15-mmsd","fpSet30-0-20-mmsd","fpSet30-0-25-
 mmsd","fpSet30-0-50-mmsd",
        "fpSet30-5-5-mmsd","fpSet30-5-10-mmsd","fpSet30-5-15-mmsd","fpSet30-5-20-mmsd","fpSet30-5-25-
 mmsd","fpSet30-5-50-mmsd",
        "fpSet30-10-5-mmsd","fpSet30-10-10-mmsd","fpSet30-10-15-mmsd","fpSet30-10-20-mmsd","fpSet30-10-
 25-mmsd","fpSet30-10-50-mmsd",
        "fpSet30-15-5-mmsd","fpSet30-15-10-mmsd","fpSet30-15-15-mmsd","fpSet30-15-20-mmsd","fpSet30-15-
 25-mmsd","fpSet30-15-50-mmsd",
        "fpSet30-20-5-mmsd","fpSet30-20-10-mmsd","fpSet30-20-15-mmsd","fpSet30-20-20-mmsd","fpSet30-20-
 25-mmsd","fpSet30-20-50-mmsd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
```

```
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("25", "50", "100", "150", "200")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_nn_fp_mmsd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.6 'matchRatesFinal_nn_is.R'

```
rm(list = ls(all = TRUE))

library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/nn_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.55, lines = 1.2)
colscaledivs <- 30

# First plot - method means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")
```

```
agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mean","indVal-0-10-mean","indVal-0-15-mean","indVal-0-20-mean","indVal-0-25-
 mean","indVal-0-50-mean",
        "indVal-5-5-mean","indVal-5-10-mean","indVal-5-15-mean","indVal-5-20-mean","indVal-5-25-
 mean","indVal-5-50-mean",
        "indVal-10-5-mean","indVal-10-10-mean","indVal-10-15-mean","indVal-10-20-mean","indVal-10-25-
 mean","indVal-10-50-mean",
        "indVal-15-5-mean","indVal-15-10-mean","indVal-15-15-mean","indVal-15-20-mean","indVal-15-25-
 mean","indVal-15-50-mean",
        "indVal-20-5-mean","indVal-20-10-mean","indVal-20-15-mean","indVal-20-20-mean","indVal-20-25-
 mean","indVal-20-50-mean"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_nn_is_means.png",
type ="png",
```

```
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Second plot - method sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-sd","indVal-0-10-sd","indVal-0-15-sd","indVal-0-20-sd","indVal-0-25-sd","indVal-0-50-sd",
        "indVal-5-5-sd","indVal-5-10-sd","indVal-5-15-sd","indVal-5-20-sd","indVal-5-25-sd","indVal-5-50-sd",
        "indVal-10-5-sd","indVal-10-10-sd","indVal-10-15-sd","indVal-10-20-sd","indVal-10-25-sd","indVal-10-50-
 sd",
        "indVal-15-5-sd","indVal-15-10-sd","indVal-15-15-sd","indVal-15-20-sd","indVal-15-25-sd","indVal-15-50-
 sd",
        "indVal-20-5-sd","indVal-20-10-sd","indVal-20-15-sd","indVal-20-20-sd","indVal-20-25-sd","indVal-20-50-
 sd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -0.5, to = 7.5, length = 17)
 # , cuts = 16
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
```

```
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/matchRatesFinal_nn_is_sd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mmxbar","indVal-0-10-mmxbar","indVal-0-15-mmxbar","indVal-0-20-mmxbar","indVal-0-25-
 mmxbar","indVal-0-50-mmxbar",
        "indVal-5-5-mmxbar","indVal-5-10-mmxbar","indVal-5-15-mmxbar","indVal-5-20-mmxbar","indVal-5-25-
 mmxbar","indVal-5-50-mmxbar",
        "indVal-10-5-mmxbar","indVal-10-10-mmxbar","indVal-10-15-mmxbar","indVal-10-20-mmxbar","indVal-10-
 25-mmxbar","indVal-10-50-mmxbar",
        "indVal-15-5-mmxbar","indVal-15-10-mmxbar","indVal-15-15-mmxbar","indVal-15-20-mmxbar","indVal-15-
 25-mmxbar","indVal-15-50-mmxbar",
        "indVal-20-5-mmxbar","indVal-20-10-mmxbar","indVal-20-15-mmxbar","indVal-20-20-mmxbar","indVal-20-
 25-mmxbar","indVal-20-50-mmxbar"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
```

```
 ),
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("25", "50", "100", "150", "200")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_nn_is_mmxbar.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
         "indVal-0-5-mmsd","indVal-0-10-mmsd","indVal-0-15-mmsd","indVal-0-20-mmsd","indVal-0-25-
 mmsd","indVal-0-50-mmsd",
         "indVal-5-5-mmsd","indVal-5-10-mmsd","indVal-5-15-mmsd","indVal-5-20-mmsd","indVal-5-25-
 mmsd","indVal-5-50-mmsd",
```

```
        "indVal-10-5-mmsd","indVal-10-10-mmsd","indVal-10-15-mmsd","indVal-10-20-mmsd","indVal-10-25-
 mmsd","indVal-10-50-mmsd",
        "indVal-15-5-mmsd","indVal-15-10-mmsd","indVal-15-15-mmsd","indVal-15-20-mmsd","indVal-15-25-
 mmsd","indVal-15-50-mmsd",
        "indVal-20-5-mmsd","indVal-20-10-mmsd","indVal-20-15-mmsd","indVal-20-20-mmsd","indVal-20-25-
 mmsd","indVal-20-50-mmsd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_nn_is_mmsd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.7 'matchRatesFinal_sdnn_fp.R'

```
rm(list = ls(all = TRUE))
```

```
library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/sdnn_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.55, lines = 1.2)
colscaledivs <- 30

# First plot - method means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mean","fpSet30-0-10-mean","fpSet30-0-15-mean","fpSet30-0-20-mean","fpSet30-0-25-
 mean","fpSet30-0-50-mean",
        "fpSet30-5-5-mean","fpSet30-5-10-mean","fpSet30-5-15-mean","fpSet30-5-20-mean","fpSet30-5-25-
 mean","fpSet30-5-50-mean",
        "fpSet30-10-5-mean","fpSet30-10-10-mean","fpSet30-10-15-mean","fpSet30-10-20-mean","fpSet30-10-25-
 mean","fpSet30-10-50-mean",
        "fpSet30-15-5-mean","fpSet30-15-10-mean","fpSet30-15-15-mean","fpSet30-15-20-mean","fpSet30-15-25-
 mean","fpSet30-15-50-mean",
        "fpSet30-20-5-mean","fpSet30-20-10-mean","fpSet30-20-15-mean","fpSet30-20-20-mean","fpSet30-20-25-
 mean","fpSet30-20-50-mean"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
```

```
 , drop = FALSE
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_fp_means.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)


# Second plot - method sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-sd","fpSet30-0-10-sd","fpSet30-0-15-sd","fpSet30-0-20-sd","fpSet30-0-25-sd","fpSet30-0-50-
 sd",
        "fpSet30-5-5-sd","fpSet30-5-10-sd","fpSet30-5-15-sd","fpSet30-5-20-sd","fpSet30-5-25-sd","fpSet30-5-50-
 sd",
        "fpSet30-10-5-sd","fpSet30-10-10-sd","fpSet30-10-15-sd","fpSet30-10-20-sd","fpSet30-10-25-sd","fpSet30-
 10-50-sd",
```

```
        "fpSet30-15-5-sd","fpSet30-15-10-sd","fpSet30-15-15-sd","fpSet30-15-20-sd","fpSet30-15-25-sd","fpSet30-
15-50-sd",
        "fpSet30-20-5-sd","fpSet30-20-10-sd","fpSet30-20-15-sd","fpSet30-20-20-sd","fpSet30-20-25-sd","fpSet30-
20-50-sd"
        )
, labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
),
, drop = FALSE
, pretty = TRUE
, par.strip.text=p.strip
, strip = strip.custom(bg = "transparent")
, contour = FALSE
, data = ag_co
, region = TRUE
# , at = seq(from = -0.5, to = 7.5, length = 17)
# , cuts = 16
, col.regions = (col = rgb.palette(colscaledivs + 1))
, xlab = "Number of plots"
, ylab = "Number of species"
, layout = c(6, 5)
, scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
, between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_fp_sd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")
```

```
agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mmxbar","fpSet30-0-10-mmxbar","fpSet30-0-15-mmxbar","fpSet30-0-20-mmxbar","fpSet30-
 0-25-mmxbar","fpSet30-0-50-mmxbar",
        "fpSet30-5-5-mmxbar","fpSet30-5-10-mmxbar","fpSet30-5-15-mmxbar","fpSet30-5-20-mmxbar","fpSet30-
 5-25-mmxbar","fpSet30-5-50-mmxbar",
        "fpSet30-10-5-mmxbar","fpSet30-10-10-mmxbar","fpSet30-10-15-mmxbar","fpSet30-10-20-
 mmxbar","fpSet30-10-25-mmxbar","fpSet30-10-50-mmxbar",
        "fpSet30-15-5-mmxbar","fpSet30-15-10-mmxbar","fpSet30-15-15-mmxbar","fpSet30-15-20-
 mmxbar","fpSet30-15-25-mmxbar","fpSet30-15-50-mmxbar",
        "fpSet30-20-5-mmxbar","fpSet30-20-10-mmxbar","fpSet30-20-15-mmxbar","fpSet30-20-20-
 mmxbar","fpSet30-20-25-mmxbar","fpSet30-20-50-mmxbar"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , drop = FALSE
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_fp_mmxbar.png",
```

```
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "fpSet30",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "fpSet30-0-5-mmsd","fpSet30-0-10-mmsd","fpSet30-0-15-mmsd","fpSet30-0-20-mmsd","fpSet30-0-25-
 mmsd","fpSet30-0-50-mmsd",
        "fpSet30-5-5-mmsd","fpSet30-5-10-mmsd","fpSet30-5-15-mmsd","fpSet30-5-20-mmsd","fpSet30-5-25-
 mmsd","fpSet30-5-50-mmsd",
        "fpSet30-10-5-mmsd","fpSet30-10-10-mmsd","fpSet30-10-15-mmsd","fpSet30-10-20-mmsd","fpSet30-10-
 25-mmsd","fpSet30-10-50-mmsd",
        "fpSet30-15-5-mmsd","fpSet30-15-10-mmsd","fpSet30-15-15-mmsd","fpSet30-15-20-mmsd","fpSet30-15-
 25-mmsd","fpSet30-15-50-mmsd",
        "fpSet30-20-5-mmsd","fpSet30-20-10-mmsd","fpSet30-20-15-mmsd","fpSet30-20-20-mmsd","fpSet30-20-
 25-mmsd","fpSet30-20-50-mmsd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , drop = FALSE
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
```

```
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         , labels = c("25", "50", "100", "150", "200")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_fp_mmsd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.8 'matchRatesFinal_sdnn_is.R'

```
rm(list = ls(all = TRUE))

library(lattice)


# There are a total of 16 multipanel plots that need to be generated.
 # 4 x method means
 # 4 x miss match means
 # 4 x method sd
 # 4 x miss match sd


# Read in the data into a generic data.frame that can be used repeatedly
unfiltered <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Data/SubSetting/Processed/sdnn_.tsv",
 header = TRUE, sep = "\t", na.strings = c("NA"))

rgb.palette <- colorRampPalette(c("grey", "brown", "yellow", "pink", "blue"), space = "Lab")
p.strip <- list(cex = 0.55, lines = 1.2)
colscaledivs <- 30

# First plot - method means for is mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")
```

```
agMatchMean <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mean", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mean","indVal-0-10-mean","indVal-0-15-mean","indVal-0-20-mean","indVal-0-25-
 mean","indVal-0-50-mean",
        "indVal-5-5-mean","indVal-5-10-mean","indVal-5-15-mean","indVal-5-20-mean","indVal-5-25-
 mean","indVal-5-50-mean",
        "indVal-10-5-mean","indVal-10-10-mean","indVal-10-15-mean","indVal-10-20-mean","indVal-10-25-
 mean","indVal-10-50-mean",
        "indVal-15-5-mean","indVal-15-10-mean","indVal-15-15-mean","indVal-15-20-mean","indVal-15-25-
 mean","indVal-15-50-mean",
        "indVal-20-5-mean","indVal-20-10-mean","indVal-20-15-mean","indVal-20-20-mean","indVal-20-25-
 mean","indVal-20-50-mean"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , drop = FALSE
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_is_means.png",
type ="png",
```

```
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)


# Second plot - method sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$match ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-sd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-sd","indVal-0-10-sd","indVal-0-15-sd","indVal-0-20-sd","indVal-0-25-sd","indVal-0-50-sd",
        "indVal-5-5-sd","indVal-5-10-sd","indVal-5-15-sd","indVal-5-20-sd","indVal-5-25-sd","indVal-5-50-sd",
        "indVal-10-5-sd","indVal-10-10-sd","indVal-10-15-sd","indVal-10-20-sd","indVal-10-25-sd","indVal-10-50-
 sd",
        "indVal-15-5-sd","indVal-15-10-sd","indVal-15-15-sd","indVal-15-20-sd","indVal-15-25-sd","indVal-15-50-
 sd",
        "indVal-20-5-sd","indVal-20-10-sd","indVal-20-15-sd","indVal-20-20-sd","indVal-20-25-sd","indVal-20-50-
 sd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , drop = FALSE
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -0.5, to = 7.5, length = 17)
 # , cuts = 16
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
```

```
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_is_sd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Third plot - miss match means for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchMean <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = mean, na.action = NULL)
names(agMatchMean) <- c("test", "sp", "pl", "match")
agMatchMean$test <- paste(agMatchMean$test, "-mmxbar", sep = "")

ag_co <- agMatchMean

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mmxbar","indVal-0-10-mmxbar","indVal-0-15-mmxbar","indVal-0-20-mmxbar","indVal-0-25-
 mmxbar","indVal-0-50-mmxbar",
        "indVal-5-5-mmxbar","indVal-5-10-mmxbar","indVal-5-15-mmxbar","indVal-5-20-mmxbar","indVal-5-25-
 mmxbar","indVal-5-50-mmxbar",
        "indVal-10-5-mmxbar","indVal-10-10-mmxbar","indVal-10-15-mmxbar","indVal-10-20-mmxbar","indVal-10-
 25-mmxbar","indVal-10-50-mmxbar",
        "indVal-15-5-mmxbar","indVal-15-10-mmxbar","indVal-15-15-mmxbar","indVal-15-20-mmxbar","indVal-15-
 25-mmxbar","indVal-15-50-mmxbar",
        "indVal-20-5-mmxbar","indVal-20-10-mmxbar","indVal-20-15-mmxbar","indVal-20-20-mmxbar","indVal-20-
 25-mmxbar","indVal-20-50-mmxbar"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
```

```
        )
 ),
 , drop = FALSE
 , pretty = TRUE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_is_mmxbar.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)



# Forth plot - miss match sd for fp mining
co <- unfiltered
coPre <- co
co <- coPre[coPre$test == "indVal",]
co$test <- paste(co$test, "-", co$n, "-", co$ss, sep = "")

agMatchSd <- aggregate(co$missmatch ~ co$test + co$sp + co$pl, data = co, FUN = sd, na.action = NULL)
names(agMatchSd) <- c("test", "sp", "pl", "match")
agMatchSd$test <- paste(agMatchSd$test, "-mmsd", sep = "")

ag_co <- agMatchSd

quartz(width = 5.9, height = 7.5)
levelplot(ag_co$match ~ as.factor(ag_co$pl) + as.factor(ag_co$sp) | ordered(ag_co$test
 , levels = c(
        "indVal-0-5-mmsd","indVal-0-10-mmsd","indVal-0-15-mmsd","indVal-0-20-mmsd","indVal-0-25-
 mmsd","indVal-0-50-mmsd",
        "indVal-5-5-mmsd","indVal-5-10-mmsd","indVal-5-15-mmsd","indVal-5-20-mmsd","indVal-5-25-
```

```
 mmsd","indVal-5-50-mmsd",
        "indVal-10-5-mmsd","indVal-10-10-mmsd","indVal-10-15-mmsd","indVal-10-20-mmsd","indVal-10-25-
 mmsd","indVal-10-50-mmsd",
        "indVal-15-5-mmsd","indVal-15-10-mmsd","indVal-15-15-mmsd","indVal-15-20-mmsd","indVal-15-25-
 mmsd","indVal-15-50-mmsd",
        "indVal-20-5-mmsd","indVal-20-10-mmsd","indVal-20-15-mmsd","indVal-20-20-mmsd","indVal-20-25-
 mmsd","indVal-20-50-mmsd"
        )
 , labels = c(
        "N=0 | Sub=5","N=0 | Sub=10","N=0 | Sub=15","N=0 | Sub=20","N=0 | Sub=25","N=0 | Sub=50",
        "N=5 | Sub=5","N=5 | Sub=10","N=5 | Sub=15","N=5 | Sub=20","N=5 | Sub=25","N=5 | Sub=50",
        "N=10 | Sub=5","N=10 | Sub=10","N=10 | Sub=15","N=10 | Sub=20","N=10 | Sub=25","N=10 | Sub=50",
        "N=15 | Sub=5","N=15 | Sub=10","N=15 | Sub=15","N=15 | Sub=20","N=15 | Sub=25","N=15 | Sub=50",
        "N=20 | Sub=5","N=20 | Sub=10","N=20 | Sub=15","N=20 | Sub=20","N=20 | Sub=25","N=20 | Sub=50"
        )
 ),
 , drop = FALSE
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = ag_co
 , region = TRUE
 # , at = seq(from = -1, to = 9, length = 21)
 # , cuts = 5
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 , layout = c(6, 5)
 , scales = list(y = list(tck = c(0.5, 0)
        , alternating = FALSE
        , cex = 0.7)
        , x = list(tck = c(0.5, 0)
        , rot=90
        , alternating = FALSE
        , labels = c("25", "50", "100", "150", "200")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25)
)

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter
 3/Figures/matchRatesFinal_sdnn_is_mmsd.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.9 SubSetTests_co_x_N.R

```
# Mac Path
```

```
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Summarized Data/SubSetTests_co.tsv",
 header = TRUE, sep = "\t")

str(d)

quartz(width = 5.9, height = 7.5)

rgb.palette <- colorRampPalette(c("grey", "red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.5, lines=1.0)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$Type
 , levels = c("0-co-5", "5-co-5", "10-co-5", "15-co-5", "20-co-5"
                        ,"0-co-10", "5-co-10", "10-co-10", "15-co-10", "20-co-10"
                        ,"0-co-15", "5-co-15", "10-co-15", "15-co-15", "20-co-15"
                        ,"0-co-20", "5-co-20", "10-co-20", "15-co-20", "20-co-20"
                        ,"0-co-25", "5-co-25", "10-co-25", "15-co-25", "20-co-25"
                        ,"0-co-50", "5-co-50", "10-co-50", "15-co-50", "20-co-50")
 , labels = c("0sp | co | Sub=5", "5sp | co | Sub=5", "10sp | co | Sub=5", "15sp | co | Sub=5", "20sp | co | Sub=5"
                        ,"0sp | co | Sub=10", "5sp | co | Sub=10", "10sp | co | Sub=10", "15sp | co | Sub=10", "20sp |
 co | Sub=10"
                        ,"0sp | co | Sub=15", "5sp | co | Sub=15", "10sp | co | Sub=15", "15sp | co | Sub=15", "20sp |
 co | Sub=15"
                        ,"0sp | co | Sub=20", "5sp | co | Sub=20", "10sp | co | Sub=20", "15sp | co | Sub=20", "20sp |
 co | Sub=20"
                        ,"0sp | co | Sub=25", "5sp | co | Sub=25", "10sp | co | Sub=25", "15sp | co | Sub=25", "20sp |
 co | Sub=25"
                        ,"0sp | co | Sub=50", "5sp | co | Sub=50", "10sp | co | Sub=50", "15sp | co | Sub=50", "20sp |
 co | Sub=50")
 )
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 # , main = "Limiting Similarity\nType II Error Rate"
 , layout = c(5, 6)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         # , labels = c("5","10","15","20","25","30","35")
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
```

```
        , rot=90
        , alternating = FALSE
        # , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "", "75", "", "150", "")
        , cex = 0.65))
 , between = list(x = 0.25, y = 0.25))

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/SubSetTests_co.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.10 SubSetTests_nn_x_N.R

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Summarized
 Data/SubSetTests_nn.tsv", header = TRUE, sep = "\t")

str(d)

quartz(width = 5.9, height = 7.5)

rgb.palette <- colorRampPalette(c("grey", "red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.5, lines=1.0)

levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$Type
 , levels = c("0-nn-5", "5-nn-5", "10-nn-5", "15-nn-5", "20-nn-5"
                        ,"0-nn-10", "5-nn-10", "10-nn-10", "15-nn-10", "20-nn-10"
                        ,"0-nn-15", "5-nn-15", "10-nn-15", "15-nn-15", "20-nn-15"
                        ,"0-nn-20", "5-nn-20", "10-nn-20", "15-nn-20", "20-nn-20"
                        ,"0-nn-25", "5-nn-25", "10-nn-25", "15-nn-25", "20-nn-25"
                        ,"0-nn-50", "5-nn-50", "10-nn-50", "15-nn-50", "20-nn-50")
 , labels = c("0sp | nn | Sub=5", "5sp | nn | Sub=5", "10sp | nn | Sub=5", "15sp | nn | Sub=5", "20sp | nn | Sub=5"
                        ,"0sp | nn | Sub=10", "5sp | nn | Sub=10", "10sp | nn | Sub=10", "15sp | nn | Sub=10", "20sp
 | nn | Sub=10"
                        ,"0sp | nn | Sub=15", "5sp | nn | Sub=15", "10sp | nn | Sub=15", "15sp | nn | Sub=15", "20sp
 | nn | Sub=15"
                        ,"0sp | nn | Sub=20", "5sp | nn | Sub=20", "10sp | nn | Sub=20", "15sp | nn | Sub=20", "20sp
 | nn | Sub=20"
                        ,"0sp | nn | Sub=25", "5sp | nn | Sub=25", "10sp | nn | Sub=25", "15sp | nn | Sub=25", "20sp
 | nn | Sub=25"
                        ,"0sp | nn | Sub=50", "5sp | nn | Sub=50", "10sp | nn | Sub=50", "15sp | nn | Sub=50", "20sp
 | nn | Sub=50")
```

```
  )
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 # , main = "Limiting Similarity\nType II Error Rate"
 , layout = c(5, 6)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         # , labels = c("5","10","15","20","25","30","35")
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         # , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "", "75", "", "150", "")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25))

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/SubSetTests_nn.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```

I.1.11 SubSetTests_sdnn_x_N.R

```
# Mac Path
rm(list = ls(all = TRUE))

library(lattice)

d <- read.table("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Summarized
 Data/SubSetTests_sdnn.tsv", header = TRUE, sep = "\t")

str(d)

quartz(width = 5.9, height = 7.5)

rgb.palette <- colorRampPalette(c("grey", "red", "yellow", "green", "blue"), space = "Lab")

colscaledivs <- 20

p.strip <- list(cex=0.5, lines=1.0)
```

```
levelplot(d$val ~ as.factor(d$Plots) * as.factor(d$Species) | ordered(d$Type
 , levels = c("0-sdnn-5", "5-sdnn-5", "10-sdnn-5", "15-sdnn-5", "20-sdnn-5"
                          ,"0-sdnn-10", "5-sdnn-10", "10-sdnn-10", "15-sdnn-10", "20-sdnn-10"
                          ,"0-sdnn-15", "5-sdnn-15", "10-sdnn-15", "15-sdnn-15", "20-sdnn-15"
                          ,"0-sdnn-20", "5-sdnn-20", "10-sdnn-20", "15-sdnn-20", "20-sdnn-20"
                          ,"0-sdnn-25", "5-sdnn-25", "10-sdnn-25", "15-sdnn-25", "20-sdnn-25"
                          ,"0-sdnn-50", "5-sdnn-50", "10-sdnn-50", "15-sdnn-50", "20-sdnn-50")
 , labels = c("0sp | sdnn | Sub=5", "5sp | sdnn | Sub=5", "10sp | sdnn | Sub=5", "15sp | sdnn | Sub=5", "20sp |
 sdnn | Sub=5"
                          ,"0sp | sdnn | Sub=10", "5sp | sdnn | Sub=10", "10sp | sdnn | Sub=10", "15sp | sdnn |
 Sub=10", "20sp | sdnn | Sub=10"
                          ,"0sp | sdnn | Sub=15", "5sp | sdnn | Sub=15", "10sp | sdnn | Sub=15", "15sp | sdnn |
 Sub=15", "20sp | sdnn | Sub=15"
                          ,"0sp | sdnn | Sub=20", "5sp | sdnn | Sub=20", "10sp | sdnn | Sub=20", "15sp | sdnn |
 Sub=20", "20sp | sdnn | Sub=20"
                          ,"0sp | sdnn | Sub=25", "5sp | sdnn | Sub=25", "10sp | sdnn | Sub=25", "15sp | sdnn |
 Sub=25", "20sp | sdnn | Sub=25"
                          ,"0sp | sdnn | Sub=50", "5sp | sdnn | Sub=50", "10sp | sdnn | Sub=50", "15sp | sdnn |
 Sub=50", "20sp | sdnn | Sub=50")
  )
 , pretty = FALSE
 , par.strip.text=p.strip
 , strip = strip.custom(bg = "transparent")
 , contour = FALSE
 , data = d
 , region = TRUE
 , at = seq(from = 0, to = 1, length = colscaledivs + 1)
 , col.regions = (col = rgb.palette(colscaledivs + 1))
 , xlab = "Number of plots"
 , ylab = "Number of species"
 # , main = "Limiting Similarity\nType II Error Rate"
 , layout = c(5, 6)
 , scales = list(y = list(tck = c(0.5, 0)
         , alternating = FALSE
         # , labels = c("5","10","15","20","25","30","35")
         , cex = 0.7)
         , x = list(tck = c(0.5, 0)
         , rot=90
         , alternating = FALSE
         # , labels = c("3", "", "5", "", "7", "", "9", "", "11", "", "13", "", "15", "", "25", "", "35", "", "75", "", "150", "")
         , cex = 0.65))
 , between = list(x = 0.25, y = 0.25))

quartz.save("/Users/MikeLavender/Google Drive/MSc Thesis/Chapter 3/Figures/SubSetTests_sdnn.png",
type ="png",
width = 5.9,
height = 7.5,
dpi = 300,
device = dev.cur()
)
```